AFATL-TR-88-125

The Image Algebra Project—Phase II

AD-A204 419

Gerhard X Ritter
Joseph N Wilson
Jennifer L Davidson

UNIVERSITY OF FLORIDA
GAINESVILLE, FL 32611

JANUARY 1989

FINAL REPORT FOR PERIOD OCTOBER 1986-JUNE 1987

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▮ United States Air Force ▮ Eglin Air Force Base, Florida

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Technical Operating Report #9	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFATL-TR-88-125
6a. NAME OF PERFORMING ORGANIZATION Dept. of Computer & Information Sciences 6b. OFFICE SYMBOL (If applicable) CIS	7a. NAME OF MONITORING ORGANIZATION Air-to-Surface Guidance Branch (AGS) Advanced Guidance Division (AG)
6c. ADDRESS (City, State, and ZIP Code) University of Florida Gainesville FL 32611	7b. ADDRESS (City, State, and ZIP Code) Air Force Armament Laboratory Eglin AFB FL 32542-5434
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFATL/AGS & DARPA/TTO 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F08635-84-C-0295
8c. ADDRESS (City, State, and ZIP Code) AFATL/AGS, Eglin AFB FL 32542-5434 DARPA/TTO, 1400 Wilson Blvd., Arlington VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62602F; PROJECT NO. 2068; TASK NO 06; WORK UNIT ACCESSION NO. 44

11. TITLE (Include Security Classification)

The Image Algebra Project - Phase II

12. PERSONAL AUTHOR(S)

Gerhard X. Ritter, Joseph N. Wilson & Jennifer L. Davidson

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Oct 86 TO Jun 87	Dec 87	142

16. SUPPLEMENTARY NOTATION

Availability of this report is specified on verso of front cover.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
16	04		Image Algebra, Mathematical Morphology, Image Processing, Edge Detection, Pattern Recognition
17	07		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this project is to develop a standard Image Algebra capable of expressing all image processing transformations, which will in turn serve as the mathematical basis for a universally acceptable image processing language. The first phase of this project focused on identifying the basic operands, operators, and description techniques which provide the mathematical structure defining the algebra. This, the second and final phase of the project, is directed at "fine tuning" the algebraic structure. The resulting advanced mathematical relationships of images, images and templates, and templates and their extended properties are consolidated in theorems and proofs.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Patrick C. Coffield	22b. TELEPHONE (Include Area Code) (904)882-2838 22c. OFFICE SYMBOL AFATL/AGS

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

PREFACE

This report covers the period from October 1986 to June 1987. It was prepared by the Department of Computer and Information Sciences, 301 Computer Science and Engineering Building, University of Florida, Gainesville, Florida 32611, under Air Force Contract F08635-84-C-0295.

The authors wish to thank the Air Force Armament Laboratory (AFATL) and the Defense Advanced Research Projects Agency (DARPA) for sponsoring this work. We are particularly grateful to Dr. Sam Lambert (AFATL), Dr. Donald Daniel (AEDC), and to Dr. Jasper Lupo (DARPA) for sensing the importance of a solidly-grounded, usable image processing algebra, and for generously sharing their resources with us. We wish to acknowledge in a special way the support received from our program manager, Mr. Neal Urquhart (ASE). His constant encouragement, helpful comments, and untiring support made the work of this project more fruitful than anyone had thought possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Explanation
$\mathbf{Z}$	the set of integers
$\mathbf{R}$	the set of real numbers
$\mathbf{C}$	the set of complex numbers
$\mathbf{Z}_{2^k}$	the set of integers with binary representation of length k
$\emptyset$	the empty set
$\in, \notin, \subset$	is an element of, is not an element of, is a subset of
$\cup, \cap$	set union, set intersection
$f:\mathbf{X}\rightarrow\mathbf{Y}$	f is a function from $\mathbf{X}$ to $\mathbf{Y}$
f^{-1}	the inverse of function f
$f\|_{\mathbf{X}}$	the restriction of f to domain $\mathbf{X}$
$\tilde{f}$	the extension of f to a larger domain
$\mathbf{X}_1\times\cdots\times\mathbf{X}_n$	the n-fold cartesian product of sets $\mathbf{X}_1, \cdots, \mathbf{X}_n$
$\prod_{i=1}^{n}\mathbf{X}_i$	the n-fold cartesian product of sets $\mathbf{X}_1, \cdots, \mathbf{X}_n$
$p_j(\mathbf{x}_1, \cdots, \mathbf{x}_n)$	projection of the jth coordinate, $\mathbf{x}_j$, of $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$
$\mathbf{R}_\infty$	the extended real number system
$\vee, \wedge$	maximum, minimum
$\{\mathbf{F}, O\}$	the algebra with value set $\mathbf{F}$ and operation set O
$\mathbf{F}$	a value set
O	a set of finitary operations
$o\in O$	o is an operation in set O
$\mathbf{X}\backslash\mathbf{Y}$	the set difference of $\mathbf{X}$ and $\mathbf{Y}$

Symbol	Explanation
$\mathbf{W}, \mathbf{X}, \mathbf{Y}$	coordinate sets
$\mathbf{w}, \mathbf{x}, \mathbf{y}$	coordinate points
$\mathbf{F}^{\mathbf{X}}$	the set of all $\mathbf{F}$ valued images on $\mathbf{X}$
$\mathbf{a}:\mathbf{F}\rightarrow\mathbf{X}$	$\mathbf{a}$ is an $\mathbf{F}$ valued image on $\mathbf{X}$
$\mathbf{a}, \mathbf{b}, \mathbf{c}$	images
$\mathbf{1}$	a constant image with values at each coordinate 1
$\mathbf{0}$	a constant image with values at each coordinate 0
$\chi_S(\mathbf{a})$	the characteristic function over set S of image $\mathbf{a}$
$f(\mathbf{a})$	the function f induced pointwise over image $\mathbf{a}$
Domain($\mathbf{a}$)	the coordinate set of image $\mathbf{a}$
Range($\mathbf{a}$)	the set of values assigned to coordinates in images $\mathbf{a}$
$\mathbf{t} = (T, t)$	the template $\mathbf{t}$
T	the configuration function of template $\mathbf{t}$
$t_y \equiv t(\mathbf{y})$	the image function of template $\mathbf{t}$
$\mathbf{r}, \mathbf{s}, \mathbf{t}$	templates
$\mathbf{F}_{\mathbf{Y}\rightarrow\mathbf{X}}$	the set of all $\mathbf{F}$ valued templates from $\mathbf{Y}$ to $\mathbf{X}$
$\oplus$	generalized convolution
$\bigotimes, \bigoslash$	multiplicative maximum, multiplicative minimum
$\boxtimes$, $\boxslash$	additive maximum, additive minimum
$\bar{\mathbf{a}}$	the complement of $\mathbf{a}$
$\mathbf{a}\|_{\mathbf{Y}}$	the restriction of $\mathbf{a}$ to the domain $\mathbf{Y}$
$\mathbf{a}\|^{(\mathbf{b},\mathbf{Y})}$	the extension of $\mathbf{a}$ to $\mathbf{b}$ on $\mathbf{Y}$
choice(S)	the choice function, returning an arbitrary element of set S

Symbol	Explanation
card(S)	the cardinality function, counting the number of elements in set S
$\sum \mathbf{a}$	the sum of all pixel values of the image **a**
$\vee \mathbf{a}$	the maximum pixel value in image **a**
$\mathbf{a}^t$	the transpose of image **a**
$/o(\mathbf{a})$	the reduce of **a** by (binary) operation o
q	the injection function $q:\mathbf{F}_k \rightarrow \prod_{i=1}^{n} \mathbf{F}_i$
q_i	the ith-coordinate injection function $q_i : \prod_{j=1}^{n} \mathbf{F}_j \rightarrow \prod_{j=1}^{n} \mathbf{F}_j$, defined by $q_i \equiv qp_i$

SECTION I
INTRODUCTION

1. BACKGROUND

The Air Force Armament Laboratory (AFATL) is developing autonomous seekers to attack both mobile land targets and fixed, high-value land targets. The purpose of the mathematical structure described in this report is to aid the efficient development of such seekers. This mathematical structure—known as the AFATL *Image Algebra*—has been specifically designed for the concise expression and clear representation of image processing and pattern recognition techniques. This structure provides a common mathematical environment for target detection, algorithm development, optimization, comparison, coding, and performance evaluation. In addition, the Image Algebra provides a mathematical basis for a universal image processing language which, when properly implemented, will greatly increase a researcher's productivity as programming tasks required to compute image transformations are greatly simplified due to the replacement of large blocks of code by concise algebraic expressions.

Several previous attempts to develop a unified algebraic approach to image processing have met only partial success in expressing all transformations of gray value images, Reference 1. In contrast, in addition to meeting the design specifications mentioned in the previous paragraph, the AFATL Image Algebra provides a complete unified algebraic structure capable of expressing all image-to-image transformations. The Image Algebra's foundation evolved from a 33-month Air Force/Defense Advanced Research Project Agency (DARPA) sponsored research effort known as the Image Algebra Project.

2. ORGANIZATION OF DOCUMENT

This report presents the accomplishments of the Image Algebra Project and provides a tutorial overview of the AFATL Image Algebra. Section I provides the background to the project. Section II provides a synopsis of the accomplishments of Phase I and Phase II by tasks. Sections III and IV describe the mathematical structure of the Image Algebra in some detail. Section V, the final section of this report, discusses Image Algebra software development.

SECTION II
PROJECT ACHIEVEMENTS

1. PHASE I ACCOMPLISHMENTS BY TASKS

The Image Algebra Project was divided into a two-phase effort with the first phase spanning 12 months and the second 21 months. The main effort of the first phase focused on identifying the basic operands and operators of the algebra and to provide a first rough cut of the overall mathematical structure defining the algebra. Specifically, the tasks and achievements of the first phase can be summarized as follows.

a. Consolidation, Classification and Description of Techniques

The task of consolidating, classifying, and describing existing image processing transforms, measurement techniques, and feature analysis techniques was completed and the results were submitted in the form of the 203-page Image Algebra technical report, Reference 2. This document collects in one place descriptions of more than 100 existing image processing procedures. From this collection, frequently occurring elemental operations were identified for use in accomplishing the next task.

b. Identification and Definition of Elemental Operations

The question as to which operations and operands ought to be regarded as elemental was addressed throughout the Phase I effort. Except for some slight refinements in definition and notation, the Phase I set of elemental operations and operands underwent no major changes during Phase II of the project. The definitions of operands and operations can be found in Section III of this report. Potential structures which include these operations were investigated for use in accomplishing the next task.

c. Identification, Description, and Evaluation of Mathematical Structures

Criteria for a set of operations and criteria for a mathematical structure were completed and reported in Reference 3. These criteria can also be found in the Phase I Final Report.

The identification, demonstration, and evaluation of image Algebra structures and properties which are based on the operations and structures already identified were initially reported upon at the first program review meeting. The Phase I Final Report also includes

this work. In addition, it evaluates a number of mathematical structures which support the Image Algebra properties.

The demonstration of the capability and versatility of the optimal Image Algebra structure was completed and reported in Reference 4. Translations of numerous image processing techniques into the Image Algebra and their expression in terms of the defined elemental operations are incorporated in the final report of Phase I.

2. PHASE II ACCOMPLISHMENTS BY TASKS

The major effort of Phase II focused on fine tuning the operands, operators, and the overall algebraic structure defined during Phase I. Development and implementation of Image Algebra software, and sponsor requested documentation were additional major required tasks. The specific Phase II tasks and their accomplishments are summarized below. The first three tasks are concerned with the mathematical development of the Image Algebra. They are closely interrelated and—from both a practical and theoretical point of view—form a coherent unit. For this reason we shall discuss their accomplishments collectively and view them as the accomplishment of one major task. Tasks numbers coincide with the sponsor's statement of work (SOW) labeling scheme.

a. Task 4.2.1: Extend Properties and Relationships

b. Task 4.2.2: Identify Principal Properties and Relationships

c. Task 4.2.3: Consolidate Theorems and Proofs.

Our efforts in accomplishing these three tasks were extremely successful. One doctoral dissertation, three masters thesis, and over 20 conference and journal articles have resulted from these efforts.

Although many basic properties, relationships and theorems were established during Phase I of the project, the main body of the final structure, consisting of a multitude of new definitions, relationships, theorems, and pertinent new insights was completed during Phase II. For example, the extension of the template definition to include parameterized templates and the extension of the single valued image algebra established during Phase I to a multidata/multisensor image algebra were all accomplished during Phase II of the project. The final derived algebraic structure not only meets all the sponsor requested requirements and specifications, but in many instances surpasses these requirements and specifications.

Sections III and IV of this report describe the structure in full detail, listing all principal properties, relationships, theorems, various applications to image processing, and the formulation of image processing techniques into the language of this structure.

d. Task 4.2.4: Summarize the Structure's Advantages and Disadvantages

We first summarize the structure's advantages. The basic advantageous properties of the algebra have been established in the professional literature and can also be ascertained from the structure's description in Sections III and IV. These properties are best summarized as follows:

(1) its basic operands are capable of modeling any image;

(2) its elemental operations can be combined to express any gray-level image transformation;

(3) its elemental operations are small in number, translucent and simple, and easily taught to potential users;

(4) its theorems enable the simplification and optimization of algorithms by means of identities involving the operations;

(5) its notation provides a deeper understanding of image manipulation operations and is capable of suggesting new techniques;

(6) its notational adaptability to programming languages allows the substitution of extremely short and concise Image Algebra expressions for equivalent blocks of code, and therefore increases programmer productivity;

(7) the (standardized) notation allows the use of libraries of transformations, which in turn lower algorithm development cost;

(8) the algebraic structure is machine independent and language independent;

(9) it is applicable to image processing as implemented on any machine, whether a massively parallel processor or a sequential processor; and

(10) the set of algebraic operations is naturally coherent. Groups of operations are interrelated by algebraic identities and inequalities. There are notions of associativity, commutativity, distributivity, inverses, identities, etc.

A highly dissonant (as opposed to coherent) set of operations would have required more elaborate notation for expressing algorithms, resulting in a loss of translucency and user-friendliness. The more naturally coherent the set of operations, the easier it is to find succinct notation for expressing these operations. In conjunction with Properties (1) and (2), it is important to note that we have been able to formally prove that this set of operations is sufficient for expressing all computable image-to-image transformations.

In regard to Property (6), a large variety of well known image processing techniques, taken from standard textbooks and journals, has been translated into the Image Algebra. Some examples of these techniques and their translations are given in this document while a more complete listing can be found in the final report of Phase I of this project. The usual formulations of these techniques are in terms of word description, illustration and/or mathematical formulae. In practice, however, a programmer would have to reformulate these descriptions and equations into a computer program which would result in far more lines of computer code than lines of description. Thus, the standard formulations, when translated into a higher level computer language, are actually far more involved than they appear. The program codes representing the simple mathematical equation of the discrete Fourier transform or Sobel edge detector are prime examples of this. In contrast, the translated versions presented in this document are, for all practical purposes, coded programs.

Image Algebra operations have been implemented in the form of a preprocessor for FORTRAN. The input to the preprocessor is a FORTRAN program that includes the Image Algebra operands and operations. The preprocessor reads such a program and converts each Image Algebra statement into a sequence of FORTRAN code that implements the Image Algebra statement. The brevity and translucency of these programs imply the attainment of at least two of the above mentioned properties, namely the increase of a programmer's productivity due to simplification and reduction of code, and the transparency of the programmer's code since each algebraic statement depicts the computation on the pixel level. For example, a Government supplied Autonomous Target Recognition (ATR) algorithm for tank detection in Forward Looking Infra Red (FLIR) images translated into the Image Algebra and implemented via the preprocessor, resulted in over 75 percent reduction of code for some of its subroutines. Moreover, a significant number of undergraduate computer science and electrical engineering majors, assigned senior year research projects concerned with developing various image processing algorithms, provided a convincing argument for Property (3). Without any previous image processing experience, these students were

performing all the usual image processing tasks such as image enhancement, edge detection, image coding, region labeling, size discrimination, etc., within the first 2 weeks into their projects. These were not canned programs, but written from scratch in terms of Image Algebra notation.

Because of its simplicity and translucency, one may doubt that the Image Algebra can provide the necessary depth and breadth that may be required by future technology. However, such doubts are easily dispelled once it is realized that one subalgebra of the full Image Algebra is isomorphic (mathematically the same) as linear algebra, the most used algebra in all the sciences. Another subalgebra of the Image Algebra includes the algebra of mathematical morphology, an area of image processing that has led to many advances in computer architectures and novel image processing techniques. Similarly, some other subalgebras of the Image Algebra have already yielded several novel image processing techniques. In particular, we have demonstrated that the Image Algebra is useful as a model of parallel image processing and as a tool for the development of parallel algorithms for computing linear image to image transforms. We have shown how the algebraic relationships resulting from the structure of the Image Algebra can provide various useful results and techniques. Thus, for instance, we have shown how the Image Algebra provides an alternative algebraic formulation of linear image processing that reflects both, contemporary as well as future computing environments, References 5 and 6. Specifically, the Image Algebra can serve as an algebraic model for linear computations on computer networks based on group structures, Cayley networks, and provide a link between these networks and existing algebraic structures. In short, many deep theoretical results and their applications to image processing technology have already been obtained. However, in view of the completeness proof of the Image Algebra, these and future results should not be all that surprising.

As to the structure's deficiencies, it should be apparent from the above discussion and also from Sections III and IV that the structure is optimal for expressing image-to-image and image-to-real or complex value transformations. However, image-to-symbolic domain transformations and symbolic domain representation and manipulation as used by image understanding developers have not been addressed in the Image Algebra. Recently, notationally cor.sistent operations and methods have been developed within the Image Algebra that can extract symbolic representations from images. The Image Algebra scheme for chain code extraction is a good example of such a method. In view of this capability, it would be desirable to develop notation that naturally unifies the image and symbolic domains. An

extension of the Image Algebra to the symbolic domain would establish a comprehensive notational standard for all computer vision tasks.

e. Task 4.2.5: Study the Feasibility of using Artificial Intelligence Techniques

In our feasibility study we found that the use of artificial intelligence techniques in conjunction with the Image Algebra can be both internal or external. Internally, for example, special templates or special template operations can be defined through the use of artificial intelligence techniques. Externally, intelligence techniques can be applied to an analysis of algorithms which are expressed in the algebra, for example, the optimization of a specific algorithm, the comparative evaluation of two algorithms, or the development of a new algorithm. Our findings were reported to the sponsor in Reference 7.

f. Task 4.2.6: Demonstration of the Algebra's Capabilities

The Algebra's capabilities have already been explained and listed in the previous subsections. Further verification of its remarkable capabilities can be ascertained from Section III. Actual demonstrations of its capabilities have been given at program review meetings and several workshops, Reference 8. In addition, a Government furnished ATR algorithm was translated into Image Algebra Fortran resulting 75 percent reduction of code and vast improvement in the understanding of coded expressions due to the translucency of Image Algebra expressions.

g. Task 4.2.7: Description/Justification of the Image Algebra as a DOD Standard

The sponsor has been provided with the documentation and justification as to why the Image Algebra should serve as a DOD image processing standard, Reference 9. Here we briefly reiterate two main reasons for justifying the Algebra as a DOD standard. In simplest terms, the Image Algebra provides for a language which, if properly implemented as a standard image processing environment, will greatly reduce research and development costs. Government savings due to commonality of language and increased productivity will dwarf any reasonable initial investment for developing the Image Algebra into a standard environment for image processing. In addition, since the foundation of this language is purely mathematical and independent upon any future computer architecture or language, the longevity of an Image Algebra Standard is assured.

Although commonality of language and cost savings are two major reasons for considering the Image Algebra as a standard language for image processing, there exist a

multitude of other reasons for desiring the broad acceptance of the Image Algebra as a component of all image processing development systems. Premier among these is the predictable influence of an Image Algebra Standard on future image processing technology. In this, it can be compared to the influence on scientific reasoning and the advancement of science due to the replacement of the myriad of different number systems (i.e. Roman, Syrian, Hebrew, Egyptian, Chinese, etc.) by the now common Indo-Arabic notation.

h. Program Management and Contractor Recommended Tasks

The remaining tasks (SOW 4.3 - 5.0) were concerned with program management such as review meeting responsibilities, cost/schedule management, reports, data and other contract required deliverables. All these were carried out to the sponsors satisfaction. In addition, a contractor recommended and sponsor approved task concerned with Image Algebra software development was initiated. Section V describes the accomplishments and results of this task.

SECTION 2

IMAGE ALGEBRA OPERANDS AND OPERATORS

1. INTRODUCTION

Throughout this document we adopt commonly accepted and, whenever possible, standardized mathematical notation. In particular, $\mathbf{Z}$, $\mathbf{R}$, $\mathbf{C}$, and $\mathbf{Z}_{2^k}$ denote the set of integers, real numbers, complex numbers, and binary numbers of fixed length k, respectively. The set theoretic notions of empty set, element of, not an element of, union, intersection, and subset will be denoted by $\emptyset$, $\in$, $\notin$, $\cup$, $\cap$, and $\subset$, respectively. We assume that the reader of this document is familiar with the basic notions of set theory. However, for sake of uniformity of notation and completeness we briefly state several basic mathematical concepts and their corresponding expressions that are used throughout the technical part of this document.

Intuitively, a function f from a set X into a set Y, written $f:X \rightarrow Y$, is a rule which assigns to each $x \in X$ some element y of Y, where the assignment of x to y by the rule f is denoted by $y = f(x)$. Throughout much of this document we shall specify functions by assignment rules and call the set $\{ (x,y) : y = f(x) \text{ and } x \in X \}$, where $f:X \rightarrow Y$, the graph of f. We will also periodically refer to certain special properties and types of functions. In particular, it will be important to distinguish between the following types of functions:

A function $f:X \rightarrow Y$ is said to be one-to-one (or 1-1) if distinct elements in X have distinct images; i.e. if $f(x) = f(z)$,then $x = z$.

A function $f:X \rightarrow Y$ is said to be onto (or f is a function from X onto Y) if every $y \in Y$ is the image of some $x \in X$; i.e. if $y \in Y$, then $\exists\ x \in X$ such that $y = f(x)$.

If $f:X \rightarrow Y$ is one-to-one and onto, then there exists a function $Y \rightarrow X$, called the inverse of f and denoted by f^{-1}, with the property that $f^{-1}(f(x)) = x \forall x \in X$.

A function $f:X \rightarrow Y$ assigning all $x \in X$ to the same single element $y \in Y$ is called a constant function.

Given a function $f: X \rightarrow Y$ and a subset $A \subset X$, then the function f considered only on A (i.e. the function $\{ (x, f(x)) : x \in A \}$ is called the restriction of f to A and is denoted by $f|_A$. Thus, $f|_A = f \cap (A \times Y)$.

In the reverse direction, if $A \subset X$ and $f: A \rightarrow Y$, then any function $\tilde{f}: X \rightarrow Y$ with the property $\tilde{f}|_A = f$, is called an extension of f over X relative to Y, which will also be denoted by $f|^X$, .i.e., $\tilde{f} = f|^X$.

Two important functions whose arguments are sets are the choice function, denoted by the word choice, and the cardinality function card. Given a set X then choice(X) returns an arbitrary element of X, i.e. choice(X)=$x \in X$. The function card on the other hand counts the number of elements in X, i.e. card(X)=n where n is either a non-negative integer or infinity.

The Cartesian product of n sets $X_1, \ldots, X_n$ is denoted by either $X_1 \times X_2 \times \cdots \times X_n$ or $\prod_{i=1}^{n} X_i$, and defined as

$$\prod_{i=1}^{n} X_i = X_1 \times X_2 \times \cdots \times X_n \equiv \{(x_1, \ldots, x_n) : x_i \in X_i \}.$$

If $x = (x_1, \ldots, x_n) \in \prod_{i=1}^{n} X_i$, then $x_i \in X_i$ is called the *i-th coordinate* of x.

Given sets $X_1, \ldots, X_n$, the function $p_j : \prod_{i=1}^{n} X_i \rightarrow X_j$, where $1 \leq j \leq n$, defined by $p_j(x_1, \cdots, x_j, \cdots, x_n) = x_j$, is called the projection onto the j-th coordinate.

Conversely, given a function $q: X \rightarrow \prod_{i=1}^{n} X_i$, then q can be written as an ordered n-tuple of functions $q = (q_1, \ldots, q_n)$, where each $q_i : X \rightarrow X_i$ is defined by $q_i = p_i q$; that is, $q_i(x) = p_i(q(x))$. The function q_i is called the *ith-coordinate function* of q.

We also found it useful to employ the extended real number system in both, image processing theory and practice. The extended real number system $\mathbf{R}_\infty$ consists of the real number system $\mathbf{R}$ to which two symbols, $+\infty$ and $-\infty$, have been adjoined. The arithmetic and logic operations of $\mathbf{R}$ are extended to $\mathbf{R}_\infty$ as follows: Let $r \in \mathbf{R}$, then

(a) $-\infty < r < \infty, \quad r \vee \infty = \infty, \quad r \wedge -\infty = -\infty, \quad r \wedge \infty = r, \quad r \vee -\infty = r$

(b) $r + \infty = +\infty \quad r - \infty = -\infty \quad \frac{r}{+\infty} = \frac{r}{-\infty} = 0$

(c) If $r > 0$, then $\quad r \cdot (+\infty) = +\infty \quad r \cdot (-\infty) = -\infty$

(d) If $r < 0$, then $\quad r \cdot (+\infty) = -\infty \quad r \cdot (-\infty) = +\infty$

The notation $\vee$ and $\wedge$ in (a) above denote maximum and minimum, respectively.

2. ALGEBRAS AND VALUE SETS

Intuitively, an algebra is simply a set together with a finite number of operations (rules) for transforming one or more elements of the set into another element of the set. In precise mathematical terms, an algebra is a system $\{ \mathbf{F}, \mathcal{O} \}$ in which $\mathbf{F}$ is a non-empty set and $\mathcal{O} = \{ o_n \}$ is a set of finitary (n-ary) operations, where each operation $o_k \in \mathcal{O}$ is a mapping

$$o_k : \prod_{i=1}^{k} \mathbf{F} \rightarrow \mathbf{F}$$

For example, if k=1 or k=2, then o_k is a unary or binary operation, respectively.

Whenever the set of operations $\mathcal{O}$ is tacitly understood it is customary to let $\mathbf{F}$ denote the algebra $\{ \mathbf{F}, \mathcal{O} \}$.

Example 2.1.

The real numbers $\mathbf{R}$ together with the operations of addition, multiplication, and maximum forms an algebra that falls into a general class of algebras known as vector space lattices.

The set $\mathbf{Z}_8 = \{ 0,1,2,....,7 \}$ with the operation of addition modulo 8 is an algebra. This specific algebra belongs to a class of algebras known as cyclic abelian groups.

Suppose $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$, where each $\mathbf{F}_i$ is an algebra whose set of finitary operations is $\mathcal{O}_i$. The set of naturally induced operations, $\mathcal{O}$, on $\mathbf{F}$ is defined as:

$$\mathcal{O} = \{ o \in \prod_{i=1}^{n} \mathcal{O}_i : \text{the coordinates of } o \text{ have the same arity } \}.$$

Thus, if $o = (o_1, ..., o_n)$ and o_i is a binary operation on $\mathbf{F}_i$ for some i between 1 and n, then

$o \in O$ only if each of the remaining coordinates o_j is a binary operation.

If $f=(f_1,.....,f_n) \in \mathbf{F}$ and $g=(g_1,.....,g_n) \in \mathbf{F}$, then for $o=(o_1,.....,o_n) \in O$ we define

$$f\, o\, g \equiv (f_1\, o_1\, g_1,.....,f_n\, o_n\, g_n)$$

The set $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$ together with its naturally induced set of operations is called a value set. It follows that every value set is an algebra.

If $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$ and $\mathbf{F}_i = \mathbf{F}_j$ for all $i,j = 1,2,....,n$, then $\mathbf{F}$ is called a homogeneous value set, otherwise $\mathbf{F}$ is called a heterogeneous value set. If $\mathbf{F}$ is homogeneous and $o=(o_1,.....,o_n)$ is an operation on $\mathbf{F}$ with $o_i = o_j$ for all $i,j = 1,2,....,n$, then o is called a homogeneous operation, otherwise o is called a heterogeneous operation. Whenever o is homogeneous, then it is customary to use o_i to denote the operation o.

Example 2.2.

For i=1,2, and 3, let $\mathbf{F}_i = \mathbf{R}$ and $O_i = \{+,*,\vee\}$. Then $\mathbf{F} = \mathbf{R}^3$ and, for example, $+=(+,+,+)$, $\vee=(\vee,\vee,\vee)$, and $o=(+,*,\vee)$ are all elements of O. Hence, if $a=(a_1,a_2,a_3)$ and $b=(b_1,b_2,b_3)$ are elements of $\mathbf{R}^3$, then $a+b=(a_1+b_1,a_2+b_2,a_3+b_3)$, $a \vee b=(a_1 \vee b_1,a_2 \vee b_2,a_3 \vee b_3)$, and $a\, o\, b=(a_1+b_1,a_2 * b_2,a_3 \vee b_3)$.

Some of the more common examples of value sets used in image processing tasks are the sets $\mathbf{Z}$, $\mathbf{R}$, $\mathbf{C}$, and $\mathbf{Z}_{2^k}$ together with their usual arithmetic and lattice operations.

Given a value set $\{ \mathbf{F}, O \}$, then in addition to the operations given by O, the image algebra of $\mathbf{F}$ valued image as defined in subsequent sections also incorporates the operations of $\cup$, $\cap$, $\setminus$, choice function, and cardinality function on subsets of $\mathbf{F}$, where $\setminus$ denotes set subtraction. Thus, if $\mathbf{F} = \mathbf{R}$, then the operations are the usual arithmetic operations $+$, $-$, $*$, $\vee$, $\wedge$, etc., together with union, intersection, subtraction, choice function, and cardinality function on subsets of $\mathbf{R}$.

3. COORDINATE SETS

For $i = 1,2,.....,k$, let $\mathbf{R}_i^{n_i}$ denote n_i-dimensional Euclidean space, each having a possibly different coordinate system assignment. Thus, for example, if $i \neq j$ but $n_i = n_j = 2$, then $\mathbf{R}_i^2$ need not have the same coordinate system as $\mathbf{R}_j^2$, even though topologically both are

equivalent to $\mathbf{R}^2$. Let $\mathbf{X}_i \subset \mathbf{R}_i^{n_i}$ be a non-empty subset of n_i-dimensional Euclidean space. A coordinate set $\mathbf{X}$ is a set of form

$$\mathbf{X} = \prod_{i=1}^{k} \mathbf{X}_i.$$

In particular, if $n_k = n$ and $k = 1$, then $\mathbf{X} \subset \mathbf{R}^n$.

The elements of $\mathbf{X}$ will be denoted by bold lower case letters. Thus, if $\mathbf{x} \in \mathbf{X} \subset \mathbf{R}^n$, then $\mathbf{x}$ is of form $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, where each coordinate x_i is a real number.

Example 2.3.

If $\mathbf{X}_1 = \{(x_1, x_2) : 1 \leq x_1 \leq 3,\ 1 \leq x_2 \leq 4,\ x_1, x_2 \in \mathbf{Z}\}$ and k=1, then $\mathbf{X} = \mathbf{X}_1 \subset \mathbf{R}^2$ is the array shown in Figure 1.

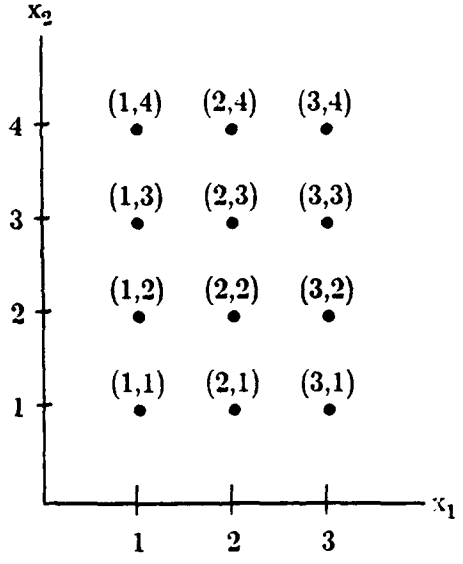


Figure 1. A Finite Rectangular Array in Euclidean 2-Space $\mathbf{R}^2$

If all the factors of $\mathbf{X}$ have identical underlying coordinate systems, then $\mathbf{X}$ is called a homogeneous coordinate set, otherwise $\mathbf{X}$ is called a heterogeneous coordinate set. In Example 2.3 above, $\mathbf{X}$ is a homogeneous coordinate set. We now provide an example of a heterogeneous coordinate set.

Example 2.4.

Let $\mathbf{X} = \mathbf{X}_1 \times \mathbf{X}_2$, where $\mathbf{X}_1$ and $\mathbf{X}_2$ denote two finite rectangular arrays in

$\mathbf{Z}^2 = \mathbf{Z} \times \mathbf{Z}$ having different underlying coordinate systems. We may visualize $\mathbf{X}_1$ and $\mathbf{X}_2$ in terms of cells as shown in Figure 2, where the center of each cell corresponds to a point in the respective array and neighboring cells correspond to neighboring points. Note the size of cells in one array differ from those in the other. This situation could correspond to two sensors of different resolving power viewing approximately the same scene.

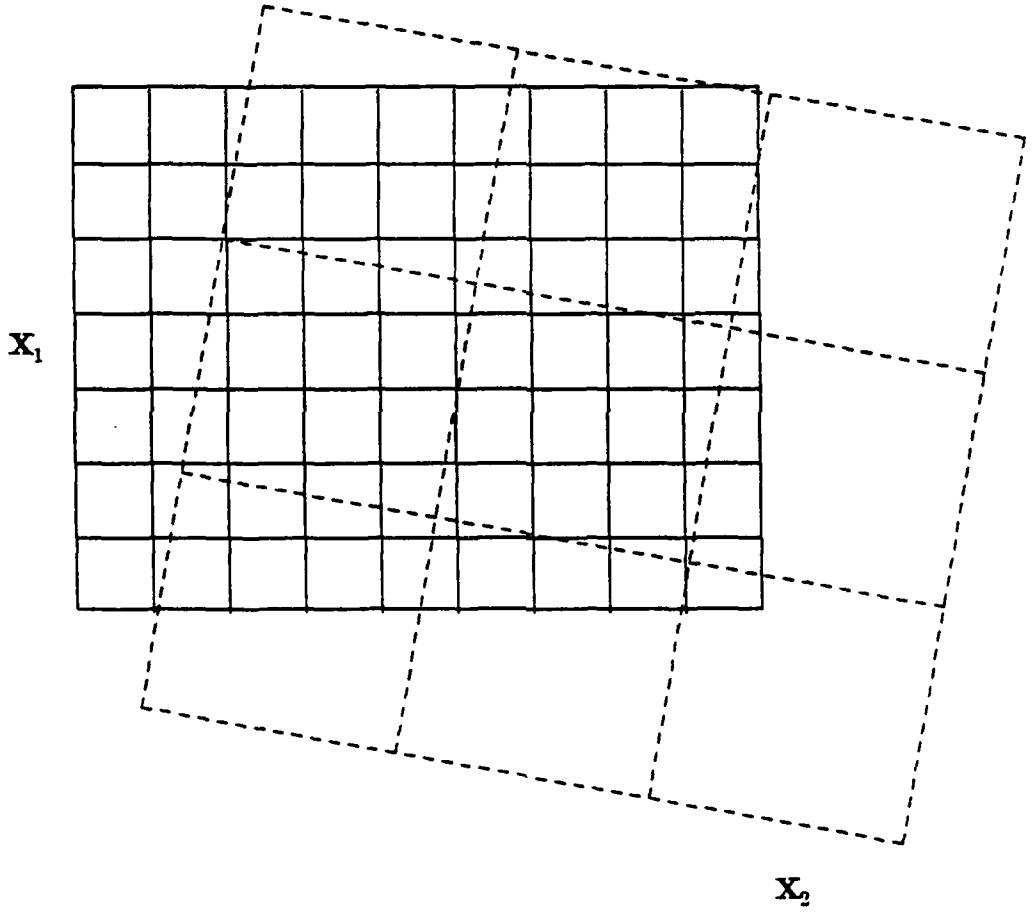


Figure 2. Two Overlapping Coordinate Sets of Different Resolution

It follows from the definition that coordinate sets can be rectangular, hexagonal, toroidal discrete arrays as well as stacks of rectangular arrays of different resolution or Euclidean

space $\mathbf{R}^n$. Providing coordinate sets with such wide varieties of shapes, sizes and dimensions allows to a coherent approach for the manipulation of differently digitized images and multisensor/multidata images.

Image algebra operations acting on coordinate sets are operations between subsets of coordinate sets as well as operations between coordinate points. In particular, operations between subsets of coordinate sets are $\cup$, $\cap$, $\setminus$, choice function, and cardinality function. Operations on coordinate sets are the usual operations between coordinate points, i.e., vector addition, scalar and vector multiplication, dot product, etc.

4. IMAGES

Thus far we have defined two types of objects, value sets and coordinate sets. These sets and their elements constitute some important operands of the image algebra. However, the most fundamental of the algebra's operands are images. The most general, yet useful, mathematical definition of an image involves the previously defined concepts of value sets and coordinate sets.

Given a coordinate and value sets $\mathbf{X}$ and $\mathbf{F}$, respectively, then an $\mathbf{F}$ valued image $\mathbf{a}$ on $\mathbf{X}$ is the graph of a function $\mathbf{a}:\mathbf{X} \rightarrow \mathbf{F}$. Thus, an $\mathbf{F}$ valued image $\mathbf{a}$ on $\mathbf{X}$ is of the form:

$$\mathbf{a} = \{ (\mathbf{x},\mathbf{a}(\mathbf{x})) : \mathbf{x} \in \mathbf{X},\ \mathbf{a}(\mathbf{x}) \in \mathbf{F} \}$$

The set $\mathbf{X}$ is called the set of image coordinates of $\mathbf{a}$, and the range of the function $\mathbf{a}$ (which is a subset of $\mathbf{F}$) is the set of image values of $\mathbf{a}$. If the value set $\mathbf{F}$ is the set of positive real numbers, then the image values of $\mathbf{a}$ are also called gray values. The set of all $\mathbf{F}$ valued images on $\mathbf{X}$ is denoted by $\mathbf{F}^{\mathbf{X}}$.

If the value set $\mathbf{F} = \mathbf{R}$ or $\mathbf{F} = \mathbf{R}^n$, then we are dealing with real valued or n-dimensional vector valued images, respectively. Similarly, replacing $\mathbf{F}$ by $\mathbf{Z}$, $\mathbf{C}$, or $\mathbf{Z}_{2^k}$, provides for integral, complex or finite digital images, respectively. Replacing $\mathbf{F}$ by $\mathbf{R}_\infty$ proves useful when manipulating raw radar images, which often contain spatial locations with out-of-range values or with no signal values (called missing values). These locations can be assigned special symbols corresponding to ∞ and $-\infty$, respectively.

The examples of the preceding paragraph should make it clear that the various choices for $\mathbf{F}$ allow for a far greater variety of image operands than are currently used by the image processing community.

5. BINARY AND UNARY OPERATIONS ON IMAGES

The elementary operations between $\mathbf{F}$ valued images are the natural induced operations of the algebraic system $\mathbf{F}$. For example, for real valued images (i.e. elements of $\mathbf{R}^{\mathbf{X}}$), the operations are the elementary operations induced by the vector lattice (= a vector space which is also a lattice) $\mathbf{R}$. Thus, the basic real valued image operations reflect the arithmetic and logic operations on $\mathbf{R}$. In particular, the binary operations of addition, multiplication, exponentiation and maximum on $\mathbf{R}^{\mathbf{X}}$ are defined as follows:

Let $\mathbf{a}, \mathbf{b} \in \mathbf{R}^{\mathbf{X}}$. Then

$$\mathbf{a} + \mathbf{b} \equiv \{ (\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) + \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \tag{1}$$

$$\mathbf{a} * \mathbf{b} \equiv \{ (\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) * \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \tag{2}$$

$$\mathbf{a} \vee \mathbf{b} \equiv \{ (\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) \vee \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \tag{3}$$

These are the basic operations for real valued images. As complex numbers are not endowed with a natural lattice structure, only operations 1 and 2 are basic operations between complex valued images. With the exception of operation 6 below, the remaining binary operations on real or complex valued images could be obtained from these basic operations either directly or in terms of series expansion.

$$\mathbf{a}^{\mathbf{b}} \equiv \{(\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x})^{\mathbf{b}(\mathbf{x})} \text{ if } \mathbf{a}(\mathbf{x}) \neq 0, \text{ else } \mathbf{c}(\mathbf{x}) = 0, \mathbf{x} \in \mathbf{X}\}. \tag{4}$$

We restrict this binary operation to those pairs of images $\mathbf{a},\mathbf{b}$ for which $\mathbf{a}(\mathbf{x})^{\mathbf{b}(\mathbf{x})} \in \mathbf{R}$ whenever $\mathbf{a}(\mathbf{x}) \neq 0$. The inverse of exponentiation is defined in the usual way by taking the logarithm. In particular, we define:

$$\log_{\mathbf{a}}\mathbf{b} \equiv \{(\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \log_{\mathbf{a}(\mathbf{x})}\mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}. \tag{5}$$

As for real numbers, $\log_{\mathbf{a}}\mathbf{b}$ is defined only for those images $\mathbf{a}$ and $\mathbf{b}$ for which $\mathbf{a}(\mathbf{x}) > 0$ and $\mathbf{b}(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathbf{X}$. The next basic binary operation, called the dot product, distinguishes itself from the above five in that its output is not an image but a real number. Let $\mathbf{X}$ be finite, then the dot product is defined as:

$$\mathbf{a} \bullet \mathbf{b} \equiv \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x})\mathbf{b}(\mathbf{x}) \tag{6}$$

An image **a** is called a constant image if the same value is assigned by **a** to each element of its coordinate set; i.e. if $\mathbf{a}(\mathbf{x}) = k$ for some real number k and for all $\mathbf{x} \in \mathbf{X}$.

Two constant images that are of utmost importance in the Image Algebra are the zero image, defined by $\mathbf{0} \equiv \{(\mathbf{x},0): \mathbf{x} \in \mathbf{X}\}$, and the unit image, defined by $\mathbf{1} \equiv \{(\mathbf{x},1): \mathbf{x} \in \mathbf{X}\}$.

Suppose $k \in \mathbf{R}$ and **a** is a constant image with $\mathbf{a}(\mathbf{x}) = k$. Then we define:

$\mathbf{b}^k \equiv \mathbf{b}^{\mathbf{a}}$ and $k^{\mathbf{b}} \equiv \mathbf{a}^{\mathbf{b}}$,

$k\mathbf{b} \equiv \mathbf{a}*\mathbf{b}$ and $k+\mathbf{b} \equiv \mathbf{a} + \mathbf{b}$,

$\log_k \mathbf{b} \equiv \log_{\mathbf{a}} \mathbf{b}$, of course $k>0$ and $\mathbf{b}(\mathbf{x})>0$ for all **x**.

We note that exponentiation is defined even when $\mathbf{a}(\mathbf{x}) = 0$. Subtraction, division and minimum are defined in terms of the basic operations and inverses. Specifically:

$\mathbf{a} - \mathbf{b} \equiv \mathbf{a} + (-\mathbf{b})$, where $-\mathbf{b} = \{(\mathbf{x},-\mathbf{b}(\mathbf{x})): (\mathbf{x},\mathbf{b}(\mathbf{x})) \in \mathbf{b}\}$,

$\mathbf{a}/\mathbf{b} \equiv \mathbf{a}*\mathbf{b}^{-1}$, and

$\mathbf{a} \wedge \mathbf{b} \equiv -(-\mathbf{a} \vee -\mathbf{b})$.

The images **0** and **1** have the obvious property $\mathbf{a}+\mathbf{0} = \mathbf{a}$ and $\mathbf{a}*\mathbf{1} = \mathbf{a}$. On the other hand, $\mathbf{b}*\mathbf{b}^{-1}$ does not necessarily equal **1**. However, $\mathbf{b}*\mathbf{b}^{-1}*\mathbf{b} = \mathbf{b}$. For this reason $\mathbf{b}^{-1}$ is called the pseudo inverse of **b**. Inequalities between images are defined in terms of maximum and minimum. Thus, for example, $\mathbf{a} \leq \mathbf{b}$ if $\mathbf{a} \vee \mathbf{b} = \mathbf{b}$. These observations show that the ring $(\mathbf{R}^{\mathbf{X}}, +, *)$ and the lattice $(\mathbf{R}^{\mathbf{X}}, \vee, \leq)$ behave very much like the ring and lattice of real numbers.

There are various useful unary operations definable in terms of the basic binary operations. For example, we have already provided the definitions of $k^{\mathbf{b}}$ and $\log_k \mathbf{b}$, the exponential of an image and the logarithm of an image **b** to the base k, respectively. In particular, the exponential of an image **b**, $\exp(\mathbf{b}) = e^{\mathbf{b}} \equiv \mathbf{a}^{\mathbf{b}}$, and the natural logarithm of **b** is defined as $\ln \mathbf{b} \equiv \log_{\mathbf{a}} \mathbf{b}$, respectively, where **a** is the constant image defined by $\mathbf{a}(\mathbf{x}) \equiv e$ for all $\mathbf{x} \in \mathbf{X}$.

If **a** and **b** are images, then the characteristic value $\chi_{>\mathbf{b}}(\mathbf{a})$ is defined as

$$\chi_{>\mathbf{b}}(\mathbf{a}) = |(\mathbf{a}-\mathbf{b}) \vee \mathbf{0}|^{-1} * |(\mathbf{a}-\mathbf{b}) \vee \mathbf{0}|$$

Thus,

$$\chi_{>\mathbf{b}}(\mathbf{a}) = \{ (\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x})=1 \text{ if } \mathbf{a}(\mathbf{x})>\mathbf{b}(\mathbf{x}), \text{ else } \mathbf{c}(\mathbf{x})=0 \}.$$

Obviously, if **a**=**b**, then $(\mathbf{a} - \mathbf{b}) \vee \mathbf{0} = \mathbf{0}$ and $\chi_{>\mathbf{b}}(\mathbf{a}) = \mathbf{0}^{-1}*\mathbf{0} = \mathbf{0}$ since by definition of exponentiation, $\mathbf{0}^{-1} = \mathbf{0}$. The complement of an image **a** is denoted by $\bar{\mathbf{a}}$ and is defined as $\bar{\mathbf{a}} = \mathbf{1} - \mathbf{a}*\mathbf{a}^{-1}$. The remaining characteristic functions of images can be defined in a similar fashion, using complementation and products. For example,

$$\chi_{\leq \mathbf{b}}(\mathbf{a}) = \overline{\chi_{>\mathbf{b}}(\mathbf{a})}$$

and

$$\chi_{\mathbf{b}}(\mathbf{a}) = \chi_{\leq \mathbf{b}}(\mathbf{a}) * \chi_{\geq \mathbf{b}}(\mathbf{a})$$

Whenever **b** is the constant image with gray values equal to k it is customary to replace **b** by k in the above definitions. Figure 3 below provides an example of the operation $\chi_{\geq \mathbf{k}}(\mathbf{a})$, where $k = 15$.

Figure 3. Example of the Operation $\chi_{\geq 15}(\mathbf{a})$

Observe that the characteristic function provides a good example as to the use of the inverse of an image and the use of pointwise multiplication and maximum operations. Another similar example is the absolute value of an image which is defined as $|\mathbf{a}| \equiv \mathbf{a} \vee (-\mathbf{a})$. These examples show that there are various image operations of different degrees of complexity that can be defined in terms of the more basic first six operations listed above. Thus, instead of listing finite strings of basic operations in order to represent the characteristic function of an image **a** with respect to any non-empty set $S \subset \mathbf{R}$, we use the more economical symbols, $\chi_S(\mathbf{a})$ and $|\mathbf{a}|$, in order to represent the image $\{ (\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = 1 \text{ if } \mathbf{a}(\mathbf{x}) \in S, \text{ else}$

$\mathbf{c}(\mathbf{x}) = 0$ } and $\mathbf{a} \vee (-\mathbf{a})$, respectively.

Thus far unary operations have been defined in terms of the basic binary operations and we observed that they are not elemental in the sense of defining the Image Algebra. The next set of unary operations could also be defined by the above named basic operations in terms of series approximation. However, this would defeat the goal of providing a simple language for image processing tasks.

The basic unary operations on $\mathbf{R}^{\mathbf{X}}$ are the functions available in most high level programming languages. In fact, any function $f: \mathbf{R} \to \mathbf{R}$ induces a function $\mathbf{R}^{\mathbf{X}} \to \mathbf{R}^{\mathbf{X}}$, again denoted by f, and defined by

$$f(\mathbf{a}) = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = f(\mathbf{a}(\mathbf{x}))\}$$

For example, $\sin(\mathbf{a}) = \{(\mathbf{x}, \sin(\mathbf{a}(\mathbf{x}))) : \mathbf{x} \in \mathbf{X}\}$.

All operations defined in this section with the exception of those involving the lattice operations $\vee$ and $\wedge$ and relations $<$ and $\leq$, can also be applied to complex valued images, i.e. elements of $\mathbf{C}^{\mathbf{X}}$. Similarly, as mentioned before, the operations on $\mathbf{F}^{\mathbf{X}}$ are the natural induced operations of the algebraic system $\mathbf{F}$. For instance, if $\mathbf{F} = \mathbf{R}^n$, then $\mathbf{a}(\mathbf{x}) \in \mathbf{R}^n$ is of form $\mathbf{a}(\mathbf{x}) = (\mathbf{a}_1(\mathbf{x}), \mathbf{a}_2(\mathbf{x}), \ldots\ldots, \mathbf{a}_n(\mathbf{x}))$, with $\mathbf{a}_i(\mathbf{x}) \in \mathbf{R}$ for i=1,2,....,n, and all previously defined operations are defined componentwise. In particular,

$$\mathbf{a}(\mathbf{x}) + \mathbf{b}(\mathbf{x}) = (\mathbf{a}_1(\mathbf{x}) + \mathbf{b}_1(\mathbf{x}), \ldots\ldots, \mathbf{a}_n(\mathbf{x}) + \mathbf{b}_n(\mathbf{x})),$$
$$\mathbf{a}(\mathbf{x}) \cdot \mathbf{b}(\mathbf{x}) = (\mathbf{a}_1(\mathbf{x}) \cdot \mathbf{b}_1(\mathbf{x}), \ldots\ldots, \mathbf{a}_n(\mathbf{x}) \cdot \mathbf{b}_n(\mathbf{x})), \text{ and}$$
$$\sin(\mathbf{a}(\mathbf{x})) = (\sin(\mathbf{a}_1(\mathbf{x})), \ldots\ldots, \sin(\mathbf{a}_n(\mathbf{x}))).$$

When considering an expression of the form $\sin(\mathbf{a})$, one rarely thinks of the image $\mathbf{a}$ as a function. However, in this paper, images are defined as functions, namely elements of $\mathbf{F}^{\mathbf{X}}$, and two important mathematical notions used in image processing are the restriction and extension of a function. We express these notions as two basic operations of the image algebra.

Let $\mathbf{a}$ be an image on $\mathbf{X}$. The restriction of $\mathbf{a}$ to a subset $\mathbf{Y}$ of $\mathbf{X}$ will be denoted by $\mathbf{a}|_{\mathbf{Y}}$. Here a user would specify the coordinate set $\mathbf{Y} \subset \mathbf{X}$. As an example, he could set $\mathbf{Y} = \{\mathbf{x} \in \mathbf{X} : 5 \leq |\mathbf{x}| \leq 20\}$. The output image $\mathbf{a}|_{\mathbf{Y}}$ is an image on $\mathbf{Y}$.

Let $\mathbf{a}$ be an image on $\mathbf{X}$, $\mathbf{b}$ an image on $\mathbf{Y}$, and $\mathbf{X} \subset \mathbf{Y}$. The extension of $\mathbf{a}$ to $\mathbf{b}$ on $\mathbf{Y}$ is defined by

$$\mathbf{a}|^{(\mathbf{b},\mathbf{Y})}(\mathbf{x}) = \begin{cases} \mathbf{a}(\mathbf{x}) \text{ if } \mathbf{x} \in \mathbf{X} \\ \mathbf{b}(\mathbf{x}) \text{ if } \mathbf{x} \in \mathbf{Y} \backslash \mathbf{X} \end{cases}$$

where the user specifies the function (image) **b** on **Y**.

Essential to the definition of a function are its domain and range. These form the remaining basic operations on images. The domain function of an image **a** is defined simply as

$$\text{Domain}(\mathbf{a}) = \text{set over which } \mathbf{a} \text{ is defined,}$$

e.g., $\text{Domain}(\mathbf{a}|_{\mathbf{Y}}) = \mathbf{Y}$. The range function of an image lies on the other side of the spectrum, it provides the set of values assigned to coordinates by the image. Thus, if **a** is an image, then

$$\text{Range}(\mathbf{a}) = \text{set of values determined by } \mathbf{a},$$

e.g., $\text{Range}(\mathbf{a}|_{\mathbf{Y}})$ is the set of all values **a** assumes on **Y**.

6. GENERALIZED TEMPLATES

In terms of image processing applications, templates and template operations are the most powerful tool of the Image Algebra. Generalized templates are the most abstract objects in the algebra. One reason for the abstraction is that our definition of template unifies and generalizes the usual concepts of templates, masks, windows and neighborhood functions of other image processing formalisms into one general mathematical entity.

Before defining generalized templates, we will look at a function that assigns to each point of a set a pair of objects. Henceforth, let **X** and **Y** be coordinate sets, **F** a value set, and let $2^{\mathbf{X}}$ denote the power set of **X**; that is, $2^{\mathbf{X}}$ is the set of all subsets of **X**. Consider a function of type

$$\mathbf{t} = (t_1, t_2) : \mathbf{Y} \rightarrow 2^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}}$$

Note that for each point $\mathbf{y} \in \mathbf{Y}$, $\mathbf{t}(\mathbf{y}) = (t_1(\mathbf{y}), t_2(\mathbf{y}))$ where $t_1(\mathbf{y}) \in 2^{\mathbf{X}}$ and $t_2(\mathbf{y}) \in \mathbf{F}^{\mathbf{X}}$. That is, $t_1(\mathbf{y}) \subset \mathbf{X}$ and $t_2(\mathbf{y})$ is an image on **X**.

For subsequent notational convenience we will denote the coordinates of this type of function in upper case script and lower case italics. In particular, the coordinates of $\mathbf{t} = (t_1, t_2)$, t_1 and t_2, will be denoted by $\mathcal{T}$ and t, respectively. In addition, we define $t_y \equiv t(\mathbf{y})$. Thus, t_y is the image $t_y = \{(\mathbf{x}, t_y(\mathbf{x})) : \mathbf{x} \in \mathbf{X}\}$.

Henceforth, all value sets under consideration are assumed to have an operation of addition and an additive identity. A generalized $\mathbf{F}$ valued template $\mathbf{t}$ from $\mathbf{Y}$ to $\mathbf{X}$ is a function $t:\mathbf{Y}\rightarrow 2^{\mathbf{X}}\times\mathbf{F}^{\mathbf{X}}$ whose coordinates (T, t) satisfy the property that for each $\mathbf{y}\in\mathbf{Y}$, $t_{\mathbf{y}}(\mathbf{x})=0$ whenever $\mathbf{x}\notin T(\mathbf{y})$. Here, 0 denotes the additive identity of $\mathbf{F}$.

An equivalent way of stating this definition is to say that an $\mathbf{F}$ valued template $\mathbf{t}$ from $\mathbf{Y}$ to $\mathbf{X}$ is a pair $\mathbf{t}=(T, t)$, where

1. $T:\mathbf{Y}\rightarrow 2^{\mathbf{X}}$; i.e., $T(\mathbf{y})\subset\mathbf{X}$ for each $\mathbf{y}\in\mathbf{Y}$ and

2. $t:\mathbf{Y}\rightarrow\mathbf{F}^{\mathbf{X}}$ with $t_{\mathbf{y}}=\{(\mathbf{x},t_{\mathbf{y}}(\mathbf{x})):\ t_{\mathbf{y}}(\mathbf{x})=0 \text{ if } \mathbf{x}\notin T(\mathbf{y}),\ \mathbf{x}\in\mathbf{X}\}$.

The set $T(\mathbf{y})$ is called the source configuration of $\mathbf{y}$; $\mathbf{y}$ is called the target point of $T(\mathbf{y})$; and the values $t_{\mathbf{y}}(\mathbf{x})$ for $\mathbf{x}\in T(\mathbf{y})$ are called the weights of $\mathbf{t}$. Figure 4 illustrates these concepts.

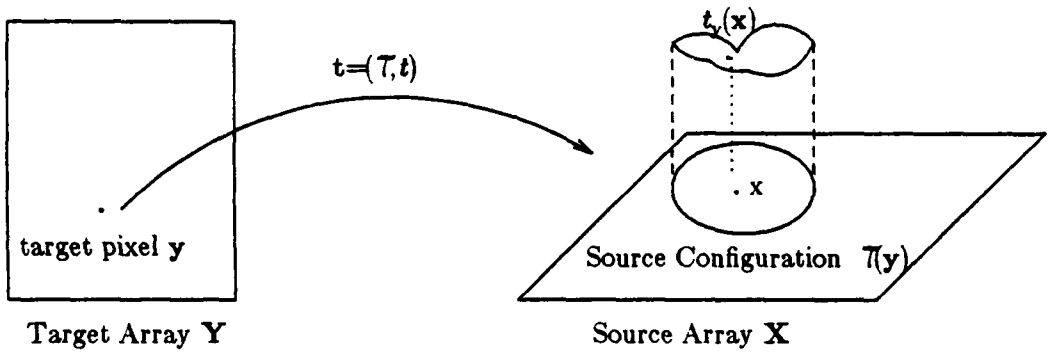


Figure 4. A Pictorial Example of a Template from $\mathbf{Y}$ to $\mathbf{X}$

The following example should further clarify the template concept. Let $\mathbf{X}\subset\mathbf{R}^2$ be a rectangular array, let $\mathbf{y}=(x,y)$ be an arbitrary point of $\mathbf{X}$, $\mathbf{x}_1=(x,y-1)$, $\mathbf{x}_2=(x+1,y)$, and $\mathbf{x}_3=(x+1,y-1)$. We now define a template $\mathbf{t}=(T,t)$ from $\mathbf{X}$ to $\mathbf{X}$ by defining - for each $\mathbf{y}\in\mathbf{X}$ - its weights as $t_{\mathbf{y}}(\mathbf{x})=0$, $t_{\mathbf{y}}(\mathbf{x}_1)=1$, $t_{\mathbf{y}}(\mathbf{x}_2)=2$, $t_{\mathbf{y}}(\mathbf{x}_3)=3$, and $t_{\mathbf{y}}(\mathbf{x})=0$ if $\mathbf{x}$ is not an element of the source configuration $T(\mathbf{y})=\{\mathbf{y},\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3\}$. Thus $\mathbf{t}$ has configuration and weights as shown:

$\mathbf{t}=$

0	2
1	3

The shaded cell in the pictorial representation of $\mathbf{t}$ indicates the location of the target point $\mathbf{y}$.

Templates are used to define those image transformations which make use of all image values within some predescribed configuration of the source or input image. For example, given an image $\mathbf{a}$ on $\mathbf{X}$ and $\mathbf{t} = (T,t)$ a template from $\mathbf{Y}$ to $\mathbf{X}$, where $\mathbf{Y}$ may be of an entirely different shape, size or dimension than $\mathbf{X}$, then an operation between $\mathbf{a}$ and $\mathbf{t}$ will transform $\mathbf{a}$ into an image $\mathbf{b}$ on $\mathbf{Y}$ where each new pixel value $\mathbf{b}(\mathbf{y})$ is computed in terms of some arithmetic and/or logic combination of the values $\mathbf{a}(\mathbf{x})$ and $t_{\mathbf{y}}(\mathbf{x})$, where $\mathbf{x} \in T(\mathbf{y})$. Initially, however, it may be convenient to view templates as masks such as the edge masks used in the Sobel or Kirsch edge detection schemes. The template configuration corresponding to a Sobel or Kirsch edge mask is always a 3x3 neighborhood configuration about the center pixel. However, the weight functions corresponding to the two Sobel edge masks (or eight Kirsch edge masks) are distinct. Thus, the two Sobel masks given in Example 2.6 are examples of two distinct templates having the same configuration. It is important to realize, however, that the notion of a template is not analogous to that of a mask. The magnification and Fourier templates presented in Example 2.10 and Section III, respectively, are examples of templates which are not masks. Here the weights or configuration changes as a function of the position of the target pixel. In addition, the target pixel of these templates does not correspond to the geometric center of the configuration. In general, given two distinct templates $\mathbf{t} = (T,t)$ and $\mathbf{s} = (S,s)$ from $\mathbf{Y}$ to $\mathbf{X}$, one, and only one, of the following three cases is possible: (i) $T \neq S$ and $t \neq s$, or (ii) $T = S$ and $t \neq s$, or (iii) $T \neq S$ and $t = s$.

The set of all $\mathbf{F}$ valued templates from $\mathbf{Y}$ to $\mathbf{X}$ will, henceforth, be denoted by $\mathbf{F}_{\mathbf{Y} \to \mathbf{X}}$.

7. OPERATIONS BETWEEN IMAGES AND TEMPLATES

There are three basic template operations that are used to transform a real valued image. They are denoted $\oplus$, $\textcircled{\vee}$, and $\boxplus$, and called generalized convolution, multiplicative maximum, and additive maximum, respectively. For complex valued images only one image-template operation is defined, namely $\oplus$.

A template operation on image $\mathbf{a}$ and template $\mathbf{t}$ computes a pixel value $\mathbf{c}(\mathbf{y})$ by performing the basic operation of addition or maximum on a weighted collection of all pixel values $\mathbf{a}(\mathbf{x})$ with coordinates $\mathbf{x} \in T(\mathbf{y})$. In particular, if $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$ and $\mathbf{t} \in \mathbf{R}_{\mathbf{Y} \to \mathbf{W}}$, where $\mathbf{X}$ and $\mathbf{W}$ are subsets of the same Euclidean space, then

$$\mathbf{a} \oplus \mathbf{t} \equiv \{(\mathbf{y}, \mathbf{c}(\mathbf{y})): \mathbf{c}(\mathbf{y}) = \sum_{\mathbf{x} \in \mathcal{T}(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) \cdot t_{\mathbf{y}}(\mathbf{x}),\ \mathbf{y} \in \mathbf{Y}\},$$

$$\mathbf{a} \boxtimes \mathbf{t} \equiv \{(\mathbf{y}, \mathbf{c}(\mathbf{y})): \mathbf{c}(\mathbf{y}) = \bigvee_{\mathbf{x} \in \mathcal{T}(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) \cdot t_{\mathbf{y}}(\mathbf{x}),\ \mathbf{y} \in \mathbf{Y}\}, \text{ and}$$

$$\mathbf{a} \boxplus \mathbf{t} \equiv \{(\mathbf{y}, \mathbf{c}(\mathbf{y})): \mathbf{c}(\mathbf{y}) = \bigvee_{\mathbf{x} \in \mathcal{T}(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x}),\ \mathbf{y} \in \mathbf{Y}\},$$

where $\sum_{\mathbf{x} \in \mathcal{T}(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) \cdot t_{\mathbf{y}}(\mathbf{x}) = 0$ whenever $\mathcal{T}(\mathbf{y}) \cap \mathbf{X} = \emptyset$. The multiplicative and additive maximums are defined only if $\mathcal{T}(\mathbf{y}) \cap \mathbf{X} \neq \emptyset$. However, if we extend $\mathbf{R}$ to $\mathbf{R}_\infty$, then $\boxtimes$ and $\boxplus$ can be defined by $\bigvee_{\mathbf{x} \in \mathcal{T}(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) \cdot t_{\mathbf{y}}(\mathbf{x}) = -\infty$ and $\bigvee_{\mathbf{x} \in \mathcal{T}(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x}) = -\infty$, respectively, whenever $\mathcal{T}(\mathbf{y}) \cap \mathbf{X} = \emptyset$.

The complementary operations of multiplicative minimum and additive minimum are defined in terms of $\boxtimes$ and $\boxplus$ as follows:

$$\mathbf{a} \boxtimes \mathbf{t} \equiv -(\mathbf{a} \boxtimes -\mathbf{t})$$

$$\mathbf{a} \boxplus \mathbf{t} \equiv -(-\mathbf{a} \boxplus -\mathbf{t})$$

In the above definitions we assume that $\mathcal{T}(\mathbf{y}) \cap \mathbf{X}$ is finite for each $\mathbf{y} \in \mathbf{Y}$. However, the definitions extend to continuous functions $\mathbf{a}(\mathbf{x})$ and $t_{\mathbf{y}}$ on compact sets $\mathcal{T}(\mathbf{y})$ and $\mathbf{X}$ with the exception that in the formulation of $\mathbf{a} \oplus \mathbf{t}$ the sum is replaced by an integral. That is,

$$\mathbf{c}(\mathbf{y}) = \int_{\mathcal{T}(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) t_{\mathbf{y}}(\mathbf{x}) d\mathbf{x}$$

Note that $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, while $\mathbf{a} \oplus \mathbf{t} \in \mathbf{R}^{\mathbf{Y}}$. Thus, template operations may be used for changing the dimensionality or size and shape of images. In particular, in addition to the usual local or global convolutions—as occur in edge enhancement, local smoothing, morphological operations and Fourier like transformations—template operations also provide a tool for image rotation, zooming, image reduction, masked extraction, and matrix multiplication. The operations can also be extended to lattice ordered vector spaces. If $\mathbf{F}_1$ is a vector space over the field $\mathbf{F}_2$, $\mathbf{t}$ an $\mathbf{F}_1$ valued template from $\mathbf{Y}$ to $\mathbf{W}$ and $\mathbf{a}$ an $\mathbf{F}_2$ valued image on $\mathbf{X}$, then $\mathbf{a} \oplus \mathbf{t}$ is an $\mathbf{F}_1$ valued image (i.e. a vector valued image) on $\mathbf{Y}$. The roles of $\mathbf{a}$ and $\mathbf{t}$ can of course be reversed, that is, $\mathbf{a}$ can be an $\mathbf{F}_1$ valued image and $\mathbf{t}$ an $\mathbf{F}_2$ valued template.

We note that other logic/arithmetic operations between image values and template values could have been defined. For instance, we could have defined

$$\mathbf{b}(\mathbf{y}) = \sum_{\mathbf{x} \in \tau(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x}) \vee t_{\mathbf{y}}(\mathbf{x}).$$

The reason for not defining the remaining logic/arithmetic operations is that some of them are easily expressible in terms of the three basic operations defined above, and no application examples could be found for the remaining combinations. This does not imply that they should henceforth be ignored. In fact, they should be kept in mind whenever new image processing techniques are being discovered that are not easily expressible within the current algebraic structure.

At first glance, the basic image/template operations may appear somewhat difficult. However, the examples provided below and those given in References 10,5,11, and 12 illustrate the inherent simplicity and power of these operations.

Example 2.5. Local Averaging

Let $\mathbf{a}$ be an image on a rectangular array $\mathbf{X} \subset \mathbf{R}^2$, $\mathbf{Y} = \mathbf{Z}^2$, and $\mathbf{t} \in \mathbf{R}_{\mathbf{Y} \to \mathbf{Y}}$ be the 3x3 neighborhood template defined as follows:

$\mathbf{t}(\mathbf{y}) =$

1	1	1
1	1	1
1	1	1

Then $\frac{1}{9}\mathbf{a} \oplus \mathbf{t}$ represents the image obtained from $\mathbf{a}$ by local averaging since the new pixel value is given by $\mathbf{c}(\mathbf{y}) = \frac{1}{9} \sum_{\mathbf{x} \in \tau(\mathbf{y}) \cap \mathbf{X}} \mathbf{a}(\mathbf{x})$.

Note that the image $\mathbf{a} \oplus \mathbf{t}$ is an image on all of $\mathbf{Z}^2$ with zero values outside of the array $\mathbf{X} \subset \mathbf{Z}^2$. Obviously, computers are not capable of storing images defined on infinite arrays. Furthermore, in practice one is only interested in the image $\frac{1}{9}(\mathbf{a} \oplus \mathbf{t})$ restricted to the array $\mathbf{X}$, that is $\frac{1}{9}(\mathbf{a} \oplus \mathbf{t})|_{\mathbf{X}}$, where $|_{\mathbf{X}}$ denotes the restriction to $\mathbf{X}$.

This problem could be solved as follows: Since $(\mathbf{a} \oplus \mathbf{t})|_X = \mathbf{a} \oplus (\mathbf{t}|_X)$, the template $\mathbf{s} \in \mathbf{R}_{X \to X}$, defined by $\mathbf{s} = \mathbf{t}|_X$ provides for the desired finite image $\frac{1}{9}(\mathbf{a} \oplus \mathbf{s})$. Thus, the question arises: "Why not simply define $\mathbf{t}$ as a template from $\mathbf{X}$ to $\mathbf{X}$ instead from $\mathbf{Z}^2$ to $\mathbf{Z}^2$?"

The reason for defining the template as we did is that this template can be used for smoothing *any* 2-dimensional image independent of its array size $\mathbf{X}$, because when defining an image $\mathbf{b}$ in a program one must declare its dimensions, i.e. the size of its underlying array (number of addresses) $\mathbf{Y}$. Hence, the program statement $\mathbf{b} = \mathbf{a} \oplus \mathbf{t}$ is equivalent to the image algebra statement $\mathbf{b} = \mathbf{a} \oplus (\mathbf{t}|_Y)$, where in a program the equal sign "=" means "replace" $\mathbf{b}$ by $\mathbf{a} \oplus \mathbf{t}$. Thus, a programmer is not faced with the task of redefining $\mathbf{t}$ for a different sized image, as would be the case if he had defined $\mathbf{t} \in \mathbf{R}_{X \to X}$ for a given $\mathbf{X}$.

Example 2.6. Sobel Edge Detection

Let $\mathbf{a}$ be an image on a rectangular array $\mathbf{X} \subset \mathbf{Z}^2$, and $\mathbf{s}_x$, $\mathbf{s}_y$ the templates shown below defined on all of $\mathbf{Z}^2$. The image algebra expression

$$[(\mathbf{a} \oplus \mathbf{s}_x)^2 + (\mathbf{a} \oplus \mathbf{s}_y)^2]^{1/2},$$

where

$\mathbf{s}_x =$

-1	0	1
-2	0	2
-1	0	1

$\mathbf{s}_y =$

-1	-2	-1
0	0	0
1	2	1

represents the Sobel edge enhanced image. Figure 5 provides an example of edge detection using the Sobel algorithm.

Figure 5. Input Image (left) and Sobel Edge Enhanced Image (right)

Example 2.7. Geometric Edge Detection

Geometric edge transforms are edge enhancement transforms that employ the image transforms $\mathbf{a} \textcircled{\vee} \mathbf{t}$ or $\mathbf{a} \boxed{\vee} \mathbf{t}$. They do not—as compared to the Sobel Transform—make use of the notion of "derivative." The following example of a geometric edge transform mimics the Sobel. Its algebraic formulation is as follows:

$$[(\mathbf{a} \textcircled{\vee}\ \mathbf{t}_1 - \mathbf{a} \textcircled{\vee}\ \mathbf{t}_2)^2 + (\mathbf{a} \textcircled{\vee}\ \mathbf{t}_3 - \mathbf{a} \textcircled{\vee}\ \mathbf{t}_4)^2]^{1/2}$$

where

$$\mathbf{t}_1 = \begin{array}{|c|c|} \hline 1 & \\ \hline 2 & 0 \\ \hline 1 & \\ \hline \end{array} \qquad \mathbf{t}_2 = \begin{array}{|c|c|} \hline & 1 \\ \hline 0 & 2 \\ \hline & 1 \\ \hline \end{array}$$

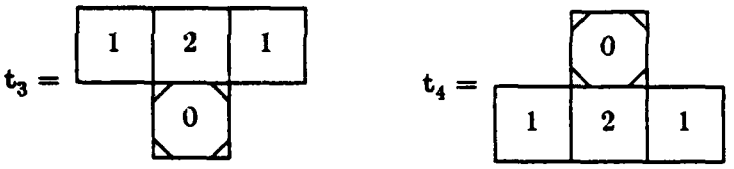


Figure 6 provides a pictorial example of this transformation.

Figure 6. Example of a Geometric Edge Transformation

Example 2.8. Dilations and Erosions

We present this example for readers familiar with the two basic notions of dilation and erosion that define all image processing schemes based on mathematical morphology. A three dimensional dilation of an image **a** by a structuring element **t** can be expressed as **a** ⊡ **t**. As before, the corresponding minimum operation ⊠ can be defined in terms of ⊡ by **a** ⊠ **t** = -(-**a** ⊡ -**t**). In particular, if **t** has only zero weights and the standard 3x3 neighborhood configuration, then **a** ⊠ **t** denotes the image obtained from **a** by replacing each pixel of **a** by the minimum of the values in its immediate neighborhood. Figure 7(b) represents the dilation **a** ⊡ **t** of the input image **a** shown in Figure 7(a), while Figure 7(c) represents the erosion of the image **a** ⊡ **t**, namely, (**a** ⊡ **t**) ⊠ **t**. The template **t** used in this example is defined as follows:

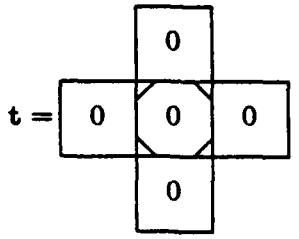


Figure 7. Template **t** of Example 2.8

(a) The Input Image **a** (b) The Dilated Image **b**=**a** ⊡ **t** (c) The Eroded Image **b** ⊡ **t**

Figure 8. Example of Dilation and Erosion

Three dimensional erosion of an image **a** by a structuring element **t** is given by the expression **a** ⊡ -**t**, since each new pixel value is given by

$$\mathbf{c}(\mathbf{x}) = \bigwedge_{\mathbf{y}\in T(\mathbf{x})} (\mathbf{a}(\mathbf{y}) - t_{\mathbf{x}}(\mathbf{y})) = \min\{\mathbf{a}(\mathbf{y}) - t_{\mathbf{x}}(\mathbf{y}) : \mathbf{y} \in T(\mathbf{y})\}.$$

It is important to note that a template may vary at different locations in both shape and weights. Thus the expression **a** ⊡ **t** may represent a far more complex algorithm than a simple dilation. In short, mathematical morphology, as used in actual image processing, is a special minor substructure of the mathematical structure represented by the Image Algebra, Reference 13.

Before discussing parameterized templates, we introduce two special but very important operations which are defined in terms of the elementary operations.

The sum of an image $\mathbf{a}$ on $\mathbf{X}$ is defined as

$$\sum \mathbf{a} \equiv |\mathbf{a}\mathbf{o}\mathbf{l}$$

and the maximum of $\mathbf{a}$ as

$$\bigvee \mathbf{a} \equiv \sum(\mathbf{a} \textcircled{\vee} \mathbf{t}),$$

where $\mathbf{t}$ is a template from $\mathbf{Y} = \{0\}$ to $\mathbf{X}$ defined by $t(0) = \mathbf{X}$ and $t_y(\mathbf{x}) = 1$ for each $\mathbf{x} \in \mathbf{X}$. Note that $\sum \mathbf{a} = \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{a}(\mathbf{x})$ is a real number and $\mathbf{a} \textcircled{\vee} \mathbf{t}$ is an image consisting of a single point.

An example that exhibits the brevity of image algebra code and involves image summation is the Euler number of a Boolean image.

Example 2.9. Euler Number

The Euler number, E, of a Boolean image is the difference between the number of objects (connected components having pixel value 1) and the number of holes in the objects. It can be computed locally, thus in real time on massively parallel architectures. The algorithm for computing E is especially short in terms of the Image Algebra:

$$E = \sum[\chi_2(\mathbf{a} \oplus \mathbf{t}) - \chi_7(\mathbf{a} \oplus \mathbf{t}) + \chi_{10}(\mathbf{a} \oplus \mathbf{t})],$$

where $\mathbf{a}$ denotes the input image and $\mathbf{t}$ the template

$\mathbf{t}(\mathbf{y}) =$

8	1
4	2

8. PARAMETERIZED TEMPLATES

Let $\mathbf{X}$, $\mathbf{Y}$ be coordinate sets, $\mathbf{F}$ a value set with identity, and P a non-empty set. A parameterized $\mathbf{F}$ valued template from $\mathbf{Y}$ to $\mathbf{X}$ with parameters in P is a function of form

$$\mathbf{t}: \mathbf{Y} \times \mathrm{P} \rightarrow 2^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}}$$

with $t(y,p) = (\mathcal{T}(y,p), t_{y,p})$ having the property that for each $(y,p) \in Y \times P$, $t_{y,p}(x) = 0$ whenever $x \notin \mathcal{T}(y,p)$. Here again we use the notation $t_{y,p} \equiv t(y,p)$.

The set P is called the set of parameters and each $p \in P$ is called a parameter for t.

Given a parameterized template $t: Y \times P \rightarrow 2^X \times F^X$, we can define a family of F valued templates from Y to X:

$$t_p : Y \rightarrow 2^X \times F^X$$

by defining for each $p \in P$

$$t_p(y) \equiv t(y,p).$$

Thus, a parameterized F valued template from Y to X gives rise to a family of regular F valued templates from Y to X, namely $\{ t_p : p \in P \}$. The following two examples should help clarify these notions.

Example 2.10. Image Magnification

Suppose $X \subset R^2$ is an $m \times n$ array, $Y = Z^2$, $P = \{ p : p=(x_0,k)$, where $x_0 \in X$, k a positive integer $\}$, and **a** an image on X. Given a pair of real numbers $r = (r_1,r_2)$, define $[r] \equiv ([r_1],[r_2])$, where $[r_i]$ denotes truncation of r_i to the nearest integer. For each $y \in Y$ and $p=(x_0,k)$, let $\mathcal{T}_p(y) = \{ x : x = [(y - x_0)/k + x_0] \}$ and $t_{y,p}(x) = 1$ if $x \in \mathcal{T}_p(y)$. Then $b = a \oplus t_p$ represents the magnification of **a** by the factor k about the point x_0. Thus, once this parameterized template has been defined, all a potential user of this template needs to supply is the magnification factor k, the point about which to magnify the image, and - in order to retain all the information - declare **b** to be of at least dimension $km \times kn$. This example also shows how a template transformation is capable of changing the size of an image. In the figure below, the image on the right represents the magnification of the image on the left by a factor of 2.

Figure 9. An Example of Image Magnification by a Factor of 2

Example 2.11. Image Rotation

Suppose $\mathbf{X} \subset \mathbf{R}^2$ is an m × n array, $\mathbf{Y} = \mathbf{Z}^2$, $P = \{p : p = (\mathbf{x}_0, \theta)$, where $\mathbf{x}_0 \in \mathbf{X}$, θ an angle between 0 and $2\pi\}$, and **a** an image on **X**. Declare **c** to be of sufficiently large dimension such that when **a** is rotated through an angle θ about a point $\mathbf{x}_0$, the rotated image fits into the array determined by **c**. We view the points of **X** and **Y** as cells, i.e., a unit box with center at each point. For each $\mathbf{y} \in \mathbf{Y}$ and $p = (\mathbf{x}_0, \theta)$, let the configuration $\mathcal{T}_p(\mathbf{y})$ be the set of cells of **X** having non-empty intersection with the cell **z**, where **z** is obtained from **y** after rotating **y** clockwise through the angle θ about the point $\mathbf{x}_0$, as shown in Figure 9.

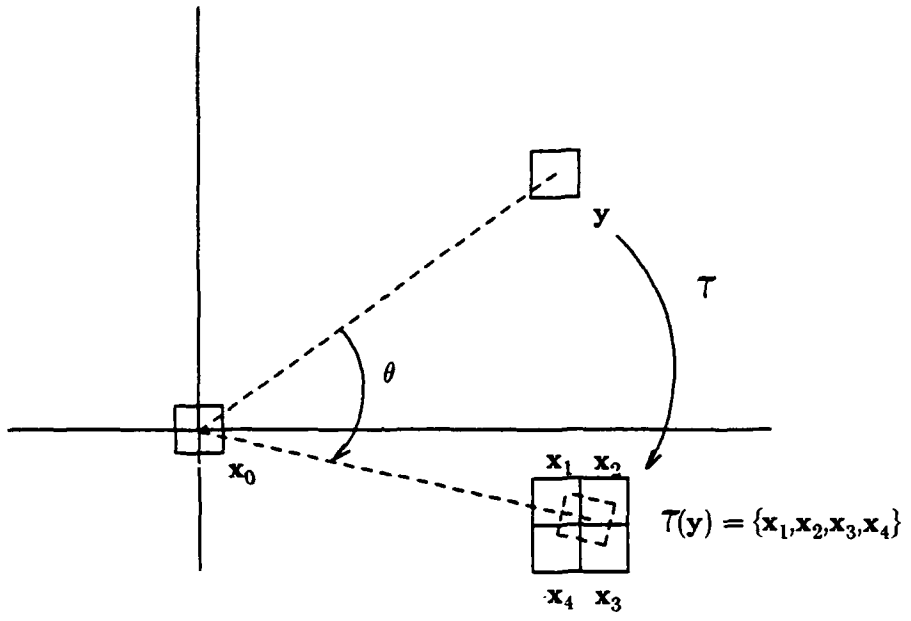


Figure 10. Target Point $\mathbf{y}$ and Source Configuration $T(\mathbf{y})$ of the Rotation Template $\mathbf{t}$

The weights $t_{y,p}(\mathbf{x}_i)$, where $\mathbf{x}_i \in T_p(\mathbf{y})$, are defined by $t_{y,p}(\mathbf{x}_i)$ = area of intersection of the preimage of $\mathbf{y}$ with $\mathbf{x}_i$. The rotated image $\mathbf{c}$ is now given by $\mathbf{c} = \mathbf{a} \oplus \mathbf{t}$, where $\mathbf{c}(\mathbf{y}) = \sum_{\mathbf{x} \in T_p(\mathbf{y})} \mathbf{a}(\mathbf{x}) t_{y,p}(\mathbf{x})$.

Figure 11 provides an example of image rotation using the template just defined.

Figure 11. Rotating an Image Through 15, 30, 45, 60, and 75 Degrees

Example 2.12. The Kirsch Edge Detector

The standard formulation of the Kirsch edge detection algorithm is to replace each pixel $\mathbf{a}(\mathbf{y})$ of the input image $\mathbf{a}$ by

$$\mathbf{c}(\mathbf{y}) = \max\{1, \max\{\,|\, 5(a_i + a_{i+1} + a_{i+2}) - 3(a_{i+3} + \ldots + a_{i+7})\,|\,: i = 1,2, \ldots, 8\,\}\},$$

where the addition of subscripts is mod 8 and the a_j's denote the following eight neighbors of $\mathbf{a}(\mathbf{y})$:

a_4	a_3	a_2
a_5	$\mathbf{a}(\mathbf{y})$	a_1
a_6	a_7	a_8

The image algebra expression representing Kirsch algorithm is given by

$$\mathbf{1} \vee \left(\bigvee_{i=1}^{8} | \mathbf{a} \oplus \mathbf{t}_i | \right),$$

where $\mathbf{1}$ denotes the unit image and $\mathbf{t}_i$ is defined as follows. For $i = 1,2, ..., 8$ define the parameterized template $\mathbf{t}_i = (\mathcal{T}_i, t_i)$ by

$$t_{i,y} = \{(\mathbf{x}, t_{i,y}(\mathbf{x}) : t_{i,y}(\mathbf{x}) = 5 \text{ if } \mathbf{x} = \mathbf{x}_i \text{ , } \mathbf{x}_{i+1} \text{ , } \mathbf{x}_{i+2} \text{ , } t_{i,y}(\mathbf{x}) = 3 \text{ if } \mathbf{x} = \mathbf{x}_{i+3} \text{ , ..., } \mathbf{x}_{i+7} \text{ , else } t_{i,y}(\mathbf{x}) = 0\} \text{ ,}$$

where the $\mathbf{x}_j$'s denote the elements of $\mathcal{T}_i(\mathbf{y})$ as shown:

$$\mathcal{T}_i(\mathbf{y}) = \begin{array}{|c|c|c|} \hline \mathbf{x}_4 & \mathbf{x}_3 & \mathbf{x}_2 \\ \hline \mathbf{x}_5 & \mathbf{y} & \mathbf{x}_1 \\ \hline \mathbf{x}_6 & \mathbf{x}_7 & \mathbf{x}_8 \\ \hline \end{array}$$

Figure 12. Example of the Kirsch Edge Detection Algorithm

9. OPERATIONS BETWEEN TEMPLATES

The operations $\oplus$, $\overset{\vee}{}$, and $\boxdot$ between images and templates generalize to operations between templates. In particular, if $\mathbf{t}$ is a real valued template from $\mathbf{Y}$ to $\mathbf{X}$ and $\mathbf{s}$ is a real valued template from $\mathbf{X}$ to $\mathbf{W}$, then we define a new template $\mathbf{r} = \mathbf{s} \oplus \mathbf{t}$ from $\mathbf{Y}$ to $\mathbf{W}$ by defining its weight function r by

$$r_y(w) = \begin{cases} \sum_{x \in T(y)} t_y(x) s_x(w), & \text{if } w \in \mathcal{R}(y) \\ 0 \text{ otherwise} \end{cases}$$

where $w \in W$ and $\mathcal{R}(y) = \bigcup_{x \in T(y)} S(x)$.

Similarly, for $r = s \boxtimes t$ we define r by

$$r_y(w) = \begin{cases} \bigvee\{ t_y(x) + s_x(w) : x \in T(y) \text{ and } w \in S(x)\} \\ \qquad \text{if } w \in \mathcal{R}(y) \\ 0 \text{ otherwise} \end{cases}$$

and for $r = s \circledvee t$ we define r by

$$r_y(w) = \begin{cases} \bigvee\{ t_y(x) s_x(w) : x \in T(y) \text{ and } w \in S(x)\} \\ \qquad \text{if } w \in \mathcal{R}(y) \\ 0 \text{ otherwise} \end{cases}$$

The complementary operations $\boxtimes$ and $\circledwedge$ are defined by

$$s \boxtimes t = -(-s \boxtimes -t)$$

$$s \circledwedge t = -(s \circledvee -t)$$

where $w \in W$ and in each case $\mathcal{R}(y) = \bigcup_{x \in T(y)} S(x)$.

If s and t are templates from Y to X, then addition, multiplication, and maximum between s and t are defined pointwise. In particular, we define

$$s + t \text{ by } (s + t)_y = s_y + t_y$$

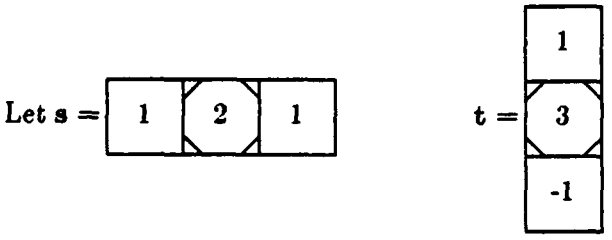
$$s * t \text{ by } (s * t)_y = s_y * t_y$$

$$\text{and } s \vee t \text{ by } (s \vee t)_y = s_y \vee t_y$$

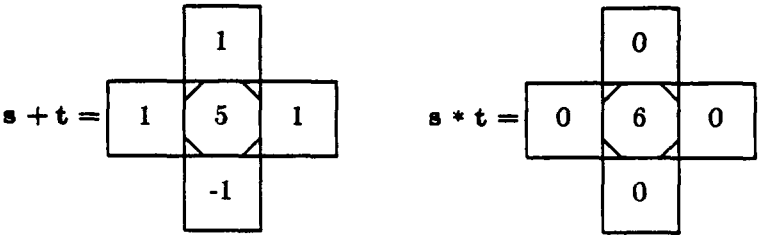
The source configuration of each of these templates at the target point y is given by $S(y) \cup T(y)$.

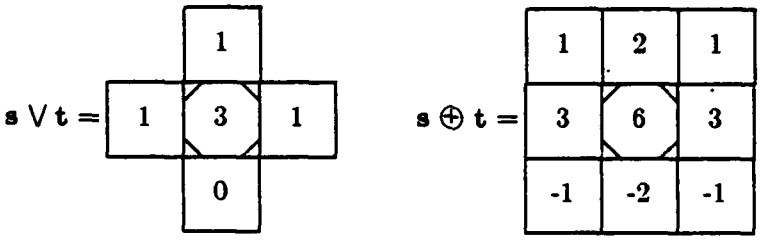
Initially, these definitions seem to be fairly complex. The following examples serve to clarify these definitions and should provide a better understanding as to how composition of templates is accomplished.

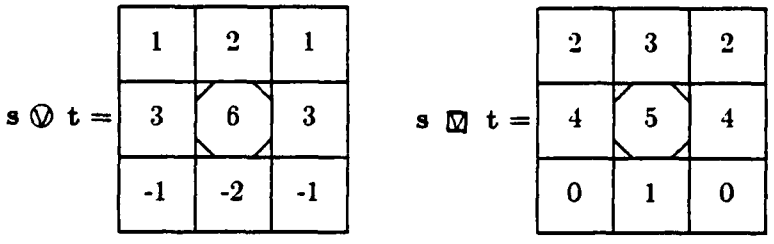
Example 2.13.



then





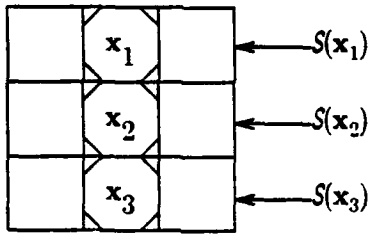


As can be seen, template addition and multiplication do not pose any great conceptual difficulties. In contrast, the operations $\mathbf{s} \oplus \mathbf{t}$, $\mathbf{s}$ ⓥ $\mathbf{t}$, and $\mathbf{s} \boxtimes \mathbf{t}$ are somewhat more intricate. For this reason we provide a step by step analysis of these operations. We start by computing the configuration $\mathcal{R}(\mathbf{y}) = \bigcup_{\mathbf{x} \in T(\mathbf{y})} S(\mathbf{x})$ at a point $\mathbf{y} \in \mathbf{Y}$. Specifically, if $\mathbf{y} \in \mathbf{Y}$, then

$\mathcal{T}(\mathbf{y})$ is of the form

$$\mathcal{T}(\mathbf{y}) = \begin{array}{|c|}\hline \mathbf{x}_1 \\ \hline \mathbf{x}_2 \\ \hline \mathbf{x}_3 \\ \hline \end{array}$$

where $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbf{X}$ with $\mathbf{y} = \mathbf{x}_2$. Hence $\mathcal{S}(\mathbf{x}_1)$, $\mathcal{S}(\mathbf{x}_2)$, and $\mathcal{S}(\mathbf{x}_3)$ are located as follows.



and the configuration is the union:

$$\mathcal{R}(\mathbf{y}) = \bigcup_{\mathbf{x} \in \mathcal{T}(\mathbf{y})} \mathcal{S}(\mathbf{x}) = \mathcal{S}(\mathbf{x}_1) \cup \mathcal{S}(\mathbf{x}_2) \cup \mathcal{S}(\mathbf{x}_3)$$

We now compute

$$r_{\mathbf{y}}(\mathbf{w}) = \begin{cases} \sum_{\mathbf{x} \in \mathcal{T}(\mathbf{y})} t_{\mathbf{y}}(\mathbf{x}) s_{\mathbf{x}}(\mathbf{w}), & \text{if } \mathbf{w} \in \mathcal{R}(\mathbf{y}) \\ 0 \text{ otherwise} \end{cases}$$

Since $r_{\mathbf{y}}(\mathbf{w}) = 0$ if $(\mathbf{w}) \notin \mathcal{R}(\mathbf{y})$, we need only compute $r_{\mathbf{y}}(\mathbf{w})$ for $\mathbf{w} \in \mathcal{R}(\mathbf{y})$. In order to illustrate this task, we relabel the points of $\mathcal{R}(\mathbf{y})$ as follows:

$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$
$\mathbf{w}_7$	$\mathbf{w}_8$	$\mathbf{w}_9$

Here $\mathbf{w}_2 = \mathbf{x}_1$, $\mathbf{w}_2 = \mathbf{x}_1$, $\mathbf{w}_5 = \mathbf{y} = \mathbf{x}_2$ and $\mathbf{w}_8 = \mathbf{x}_3$. Computation of $r_{\mathbf{y}}(\mathbf{w}_i)$ for each $i = 1, \ldots, 9$ now follows from the definition:

$$r_y(w_i) = \sum_{x \in T(y)} t_y(x) s_x(w_i), \text{ where } w_i \in S(x)$$

Specifically,

$$r_y(w_1) = t_y(x_1) s_{x_1}(w_1) = 1*1 = 1$$

$$r_y(w_2) = t_y(x_1) s_{x_1}(w_2) = 1*2 = 2$$

$$r_y(w_3) = t_y(x_1) s_{x_1}(w_3) = 1*1 = 1$$

$$r_y(w_4) = t_y(x_2) s_{x_2}(w_4) = 3*1 = 3$$

$$r_y(w_5) = t_y(x_2) s_{x_2}(w_5) = 3*2 = 6$$

$$r_y(w_6) = t_y(x_2) s_{x_2}(w_6) = 3*1 = 3$$

$$r_y(w_7) = t_y(x_3) s_{x_3}(w_7) = -1*1 = -1$$

$$r_y(w_8) = t_y(x_3) s_{x_3}(w_8) = -1*2 = -2$$

$$r_y(w_9) = t_y(x_3) s_{x_3}(w_9) = -1*1 = -1$$

Note that since $w_1 \notin S(x_2) \cup S(x_3)$, the definition does not require that the quantities $s_{x_2}(w_1)$ and $s_{x_3}(w_1)$ enter into the computation of w_1. However, since $s_{x_2}(w_1)$ and $s_{x_3}(w_1)$ are both zero, there is no harm in including them in the sum. In fact for the operation $\oplus$ it is always possible to compute w_i by relaxing the condition stated below the $\sum$ and including extra terms, all of which equal zero. Thus

$$r_y(w_i) = \sum_{x \in T(y)} t_y(x) s_x(w_i)$$

$$= \sum_{j=1}^{3} t_y(x_j) s_{x_j}(w_i)$$

$$= t_y(x_1) s_{x_1}(w_i) + t_y(x_2) s_{x_2}(w_i) + t_y(x_3) s_{x_3}(w_i)$$

Therefore one may alternatively compute $r_y(w_1)$, etc. as follows.

$$r_y(w_1) = t_y(x_1)s_{x_1}(w_1) + t_y(x_2)s_{x_2}(w_1) + t_y(x_3)s_{x_3}(w_1) = 1*1 + 3*0 + (-1)*0 = 1$$

since $t_y(x_1) = 1$, $t_y(x_2) = 3$, and $t_y(x_3) = -1$. And similarly, for i = 2, 3 and 4,

$$r_y(w_2) = t_y(x_1)s_{x_1}(w_2) + t_y(x_2)s_{x_2}(w_2) + t_y(x_3)s_{x_3}(w_2) = 1*2 + 3*0 + (-1)*0 = 2$$

$$r_y(w_3) = t_y(x_1)s_{x_1}(w_3) + t_y(x_2)s_{x_2}(w_3) + t_y(x_3)s_{x_3}(w_3) = 1*1 + 3*0 + (-1)*0 = 1$$

$$r_y(w_4) = t_y(x_1)s_{x_1}(w_4) + t_y(x_2)s_{x_2}(w_4) + t_y(x_3)s_{x_3}(w_4) = 1*0 + 3*1 + (-1)*0 = 3$$

Continuing in this fashion, we obtain $r_y(w_5) = 6$, $r_y(w_6) = 3$, $r_y(w_7) = -1$, $r_y(w_8) = -2$, and $r_y(w_9) = -1$.

In order to determine $s \boxtimes t$, we need to compute for i = 1,...,9:

$$r_y(w_i) = \bigvee_{x \in T(y)} \{ t_y(x) + s_x(w_i) : w_i \in S(x) \}$$

Equivalently, $r_y(w_i) =$

$$\{t_y(x_1) + s_{x_1}(w_i) : w_i \in S(x_1)\} \vee \{t_y(x_2) + s_{x_2}(w_i) : w_i \in S(x_2)\} \vee \{t_y(x_3) + s_{x_3}(w_i) : w_i \in S(x_3)\}.$$

Now w_1, w_2, w_3 are elements of $S(x_1)$, but none is an element of either $S(x_2)$ or $S(x_3)$. Thus

$$r_y(w_1) = t_y(x_1) + s_{x_1}(w)_1) = 1+1 = 2$$

$$r_y(w_2) = t_y(x_1) + s_{x_1}(w)_2) = 1+2 = 3$$

$$r_y(w_3) = t_y(x_1) + s_{x_1}(w)_3) = 1+1 = 2$$

Similarly, since w_4, w_5, $w_6 \in S(x_2)$ and w_7, w_8, $w_9 \in S(x_3)$, it follows that

$$r_y(w_4) = t_y(x_2) + s_{x_2}(w)_4) = 3+1 = 4$$

$$r_y(w_5) = t_y(x_2) + s_{x_2}(w)_5) = 3+2 = 5$$

$$r_y(w_6) = t_y(x_2) + s_{x_2}(w)_6) = 3+1 = 4$$

$$r_y(w_7) = t_y(x_2) + s_{x_2}(w)_7) = -1+1 = 0$$

$$r_y(w_8) = t_y(x_3) + s_{x_3}(w)_8) = -1+2 = 1$$

$$r_y(w_9) = t_y(x_3) + s_{x_3}(w)_9) = -1+1 = 0$$

For $s \textcircled{\scriptsize V} t$ we need only replace the + sign in the definition of $\boxed{\text{V}}$ by multiplication. Specifically,

$$r_y(w_1) = t_y(x_1) s_{x_1}(w_1) = 1*1 = 1$$

$$r_y(w_2) = t_y(x_1) s_{x_1}(w_2) = 1*2 = 2$$

Similarly for $r_y(w_i)$ with $i = 3,..,9$.

We observe that in this particular example, we have the coincidence that $s \oplus t = t \textcircled{\scriptsize V} s$. The reason for this is that for each distinct pair $x,x' \in T(y)$, $S(x) \cap S(x') = \emptyset$. If, for example, $T(y)$ were a von Neumann or cruciform configuration, then equality need not hold.

Subtraction, division, minimum, scalar multiplication -- etc., can be defined from these basic operations in a straight forward manner. Thus, for example, $t - s$ is defined by $(t-s)_y = t_y - s_y$.

Template composition and decomposition are the primary reason for introducing operations between generalized templates. Composition and decomposition of templates provide a tool for algorithm optimization. For instance, if s and t are as in the example above and $r = s \oplus t$, then computation of $a \oplus r = a \oplus (s \oplus t)$ by $(a \oplus s) \oplus t$ uses 6 local multiplications instead of 9.

In general, if r is an nxn template, and s and t are decompositions of r into $1 \times n$ and $n \times 1$ templates, respectively, then the computation of $a \oplus r$ by $(a \oplus s) \oplus t$ uses 2n multiplications instead of n^2. General methods for template decomposition and applications of decompositions to algorithm optimization can be found in Reference 5 and in Section III.

10. MULTIVALUE, MULTIDATA AND MULTISENSOR IMAGES

Let a be an F valued image on X. If X is homogeneous and $F = \prod_{i=1}^{n} F_i$ is homogeneous with $n > 1$, then a is called a multivalue image. If F is heterogeneous, then a is called a multidata image.

EXAMPLE 2.14.

A LANDSAT image, $\mathbf{a}$, of n-spectral bands provides an example of a multivalue image. Here $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i = \mathbf{R}^n$ and $a(\mathbf{x}) = (a_1(\mathbf{x}),.....,a_n(\mathbf{x}))$, with each $a_i(\mathbf{x}) \in \mathbf{R}$.

If $\mathbf{X}$ is heterogeneous and $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$ is homogeneous with $n > 1$, then $\mathbf{a}$ is called a multisensor/multivalue image. If both $\mathbf{X}$ and $\mathbf{F}$ are heterogeneous, then $\mathbf{a}$ is called a multisensor/multidata image. If $\mathbf{X}$ is heterogeneous and the underlying value set $\mathbf{F}$ is known, then $\mathbf{a}$ is simply called a multisensor image.

If $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$, and $\mathbf{a}$ is a multivalue or multidata image on $\mathbf{X}$ with values in $\mathbf{F}$, then $\mathbf{a}$ can always be viewed as a stack of n images $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n)$, where the i-th coordinate image $\mathbf{a}_i$ of the stack is defined as

$$\mathbf{a}_i = p_i(\mathbf{a}) = \{(\mathbf{x}, a_i(\mathbf{x})) : a_i(\mathbf{x}) = p_i(a(\mathbf{x}))\}.$$

Multisensor images are special cases of $\mathbf{F}$ valued images on $\mathbf{X}$ viewed from a different perspective. For example, let $\mathbf{X} = \mathbf{X}_1 \times \mathbf{X}_2$, where $\mathbf{X}_1$ and $\mathbf{X}_2$ denote two finite rectangular arrays in $\mathbf{Z}^2$ shown in Example 2.4. Now suppose that $\mathbf{a}_1 : \mathbf{X}_1 \rightarrow \mathbf{F}_1$ defines an $\mathbf{F}_1$ valued image on $\mathbf{X}_1$, and $\mathbf{a}_2 : \mathbf{X}_2 \rightarrow \mathbf{F}_2$ defines an $\mathbf{F}_2$ valued image on $\mathbf{X}_2$; i.e. let

$$\mathbf{a}_1 = \{(\mathbf{x}_1, \mathbf{a}_1(\mathbf{x}_1)) : \mathbf{x}_1 \in \mathbf{X}_1\}, \text{ and } \mathbf{a}_2 = \{(\mathbf{x}_2, \mathbf{a}_2(\mathbf{x}_2)) : \mathbf{x}_2 \in \mathbf{X}_2\}$$

Then the two images, $\mathbf{a}_1$ and $\mathbf{a}_2$, viewed as an ordered pair, define an $\mathbf{F} = \mathbf{F}_1 \times \mathbf{F}_2$ valued image $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2)$ on $\mathbf{X} = \mathbf{X}_1 \times \mathbf{X}_2$ if we define

$$\mathbf{a}: \mathbf{X} \rightarrow \mathbf{F} \text{ by } \mathbf{a}(\mathbf{x}) = \mathbf{a}(\mathbf{x}_1, \mathbf{x}_2) \equiv (\mathbf{a}_1(\mathbf{x}_1), \mathbf{a}_2(\mathbf{x}_2))$$

That is,

$$\mathbf{a} = \{(\mathbf{x}, \mathbf{a}(\mathbf{x})) : \mathbf{x} \in \mathbf{X}\} \equiv \{((\mathbf{x}_1, \mathbf{x}_2), (\mathbf{a}_1(\mathbf{x}_1), \mathbf{a}_2(\mathbf{x}_2)) : \mathbf{x}_1 \in \mathbf{X}_1, \mathbf{x}_2 \in \mathbf{X}_2\}.$$

In general, if i=1,2,...,n, and $\mathbf{a}_i \in \mathbf{F}_i^{\mathbf{X}_i}$, that is, $\mathbf{a}_i = \{(\mathbf{x}_i, \mathbf{a}_i(\mathbf{x}_i)) : \mathbf{x}_i \in \mathbf{X}_i\}$, then the image $\mathbf{a}$ defined as

$$\mathbf{a} = \{(\mathbf{x}, \mathbf{a}(\mathbf{x})) : \mathbf{a}(\mathbf{x}) = ((\mathbf{a}_1(\mathbf{x}_1), \mathbf{a}_2(\mathbf{x}_2)), \cdots, \mathbf{a}_n(\mathbf{x}_n))\}$$

is an element of $\mathbf{F}^{\mathbf{X}}$, where $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$ and $\mathbf{X} = \prod_{i=1}^{n} \mathbf{X}_i$. Observe that

$$\mathbf{F}^{\mathbf{X}} = (\prod_{i=1}^{n} \mathbf{F}_i)^{\mathbf{X}} = (\prod_{i=1}^{n} \mathbf{F}_i)^{(\prod_{i=1}^{n} \mathbf{X}_i)} = \mathbf{F}^{(\prod_{i=1}^{n} \mathbf{X}_i)}.$$

However, the special kind of multisensor image just described can also be viewed as an element of $\prod_{i=1}^{n} (\mathbf{F}_i)^{X_i}$ since it can be written in the form $\mathbf{a}=(\mathbf{a}_1,\mathbf{a}_2,.....,\mathbf{a}_n)$, where $\mathbf{a}_i \in \mathbf{F}_i^{X_i}$. We note that each $\mathbf{a}_i$ may again be a multidata or multivalue image as no restriction was imposed on the value set $\mathbf{F}_i$.

11. MULTIVALUE, MULTIDATA AND MULTISENSOR IMAGE OPERATIONS

The unary and binary operations on multivalue/multidata and multisensor images are the operations induced by the set $\mathcal{O}$ of operations of the underlying value set $\mathbf{F}$. For instance, if $\mathbf{F}=\prod_{i=1}^{n} \mathbf{F}_i$, $\mathbf{a}=\{(\mathbf{x},\mathbf{a}(\mathbf{x})) : \mathbf{a}(\mathbf{x})=(\mathbf{a}_1(\mathbf{x}),.....,\mathbf{a}_n(\mathbf{x}))\}$ and $\mathbf{b}=\{(\mathbf{x},\mathbf{b}(\mathbf{x})) : \mathbf{b}(\mathbf{x})=(\mathbf{b}_1(\mathbf{x}),.....,\mathbf{b}_n(\mathbf{x}))\}$ are $\mathbf{F}$ valued images on $\mathbf{X}$, and $o=(o_1,.....,o_n)\in\mathcal{O}$ a binary operation on $\mathbf{F}$, then

$$\mathbf{a}\,o\,\mathbf{b}=\{(\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x})=(\mathbf{a}_1(\mathbf{x})\,o_1\,\mathbf{b}_1(\mathbf{x}),.....,\mathbf{a}_n(\mathbf{x})\,o_n\,\mathbf{b}_n(\mathbf{x}))\}$$

or, equivalently,

$$\mathbf{a}\,o\,\mathbf{b}\equiv(\mathbf{a}_1\,o_1\,\mathbf{b}_1,.....,\mathbf{a}_n\,o_n\,\mathbf{b}_n).$$

If o is unary, then

$$o(\mathbf{a})=\{(\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x})=(o_1(\mathbf{a}_1(\mathbf{x})),.....,o_n(\mathbf{a}_n(\mathbf{x})))\}.$$

or, equivalently,

$$o(\mathbf{a})=(o_1(\mathbf{a}_1),.....,o_n(\mathbf{a}_n)).$$

Example 2.15.

If $\mathbf{F}=\mathbf{R}^n$ and o is a binary homogeneous operation with $o_i=+$, then

$$\mathbf{a}+\mathbf{b}\equiv(\mathbf{a}_1+\mathbf{b}_1,.....,\mathbf{a}_n+\mathbf{b}_n)=\{(\mathbf{x},\mathbf{c}(\mathbf{x}) : \mathbf{c}(\mathbf{x})=(\mathbf{a}_1(\mathbf{x})+\mathbf{b}_1(\mathbf{x}),.....,\mathbf{a}_n(\mathbf{x})+\mathbf{b}_n(\mathbf{x}))\}$$

Similarly, if o is a unary homogeneous operation with $o_i=\sin$, then

$$\sin(\mathbf{a})=(\sin(\mathbf{a}_1),.....,\sin(\mathbf{a}_n))=\{(\mathbf{x},\mathbf{c}(\mathbf{x}) : \mathbf{c}(\mathbf{x})=(\sin(\mathbf{a}_1(\mathbf{x})),.....,\sin(\mathbf{a}_n(\mathbf{x})))\}.$$

and if $o_i = \chi_{\geq}$, then

$$\chi_{\geq\mathbf{b}}(\mathbf{a}) = \{(\mathbf{x},\mathbf{c}(\mathbf{x})) : \mathbf{c}_i(\mathbf{x}) = 1 \text{ if } \mathbf{a}_i(\mathbf{x}) \geq \mathbf{b}_i(\mathbf{x}), \text{ otherwise } \mathbf{c}_i(\mathbf{x}) = 0,\ \mathbf{x}\in\mathbf{X}\}$$

If $\mathbf{F}=\mathbf{R}^n$ as in Example 2.15, then the operations defined for $\mathbf{R}$ valued images extend in a natural way to $\mathbf{R}^n$ valued images. For example, if $\mathbf{a}$ and $\mathbf{b}$ are $\mathbf{R}^n$ valued images on $\mathbf{X}$, then the dot product is defined as

$$\mathbf{a}\bullet\mathbf{b} = \sum_{\mathbf{x}\in\mathbf{X}} \mathbf{a}(\mathbf{x})\mathbf{b}^t(\mathbf{x})$$

where $\mathbf{b}^t(\mathbf{x})$ denotes the transpose of $\mathbf{b}(\mathbf{x})$. The notion of scalar multiplication and addition are extended to vector multiplication and vector addition in a similar fashion. In particular, if $\mathbf{b}=\{(\mathbf{x},\mathbf{b}(\mathbf{x}):\forall\mathbf{x}\in\mathbf{X},\ \mathbf{b}_i(\mathbf{x})=k_i\}$, then $\mathbf{b}$ can be viewed as a vector $\mathbf{b}=\mathbf{k}=(k_1,\ldots,k_n)\in\mathbf{R}^n$ and we define

$$\mathbf{ka}\equiv\mathbf{b}*\mathbf{a}=\{(\mathbf{x},\mathbf{c}(\mathbf{x})):\mathbf{c}_i(\mathbf{x})=k_i\mathbf{a}_i(\mathbf{x}),\ \mathbf{x}\in\mathbf{X}\}.$$

and

$$\mathbf{k}+\mathbf{a}\equiv\mathbf{b}+\mathbf{a}=\{(\mathbf{x},\mathbf{c}(\mathbf{x})):\mathbf{c}_i(\mathbf{x})=k_i+\mathbf{a}_i(\mathbf{x}),\ \mathbf{x}\in\mathbf{X}\}$$

Thus, if $\mathbf{b}$ is the constant vector valued image k, i.e, $\mathbf{b}=\mathbf{k}=(k,\ldots,k)$, then again,

$$\mathbf{b}*\mathbf{a}\equiv\mathbf{va}\equiv k\mathbf{a}\quad\text{and}$$

$$\mathbf{b}+\mathbf{a}\equiv\mathbf{k}+\mathbf{a}\equiv k+\mathbf{a}$$

Obviously, these notions extend to $\mathbf{F}=\mathbf{C}^n$ valued images and vector valued images in general. In comparison to the next set of operations, the operations just discussed are not new operations, but simply the naturally induced operations of $\{\mathbf{F},\mathcal{O}\}$.

Let $\mathbf{F}=\prod_{i=1}^{k}\mathbf{F}_i$ and $\mathbf{F}'=\prod_{i=1}^{n}\mathbf{F}'_i$ be two value sets. If $k<n$, then any function $g:\mathbf{F}\rightarrow\mathbf{F}'$ is called a data splitting function. If $k>n$, then g is called a data fusion function. In either case, g is of the form $g=(g_1,g_2,\ldots,g_n)$ where $g_i=p_ig$.

Example 2.16.

The function g: $\mathbf{R}\rightarrow\mathbf{R}^2$ is defined by $g(r)=(\cos r,\sin r)$ is an example of a value splitting function. The function f: $\mathbf{R}^2\rightarrow\mathbf{R}$ defined by $f(r_1,r_2)=r_1+r_2$ provides an example of a data fusion function.

One of the most common data fusion operations is the reduce operation. Given a homogeneous value set $\mathbf{F}=\prod_{i=1}^{n}\mathbf{F}_i$, $\mathbf{a}=(\mathbf{a}_1,\ldots,\mathbf{a}_n)$ an $\mathbf{F}$ valued image, and $o=(o_1,\ldots\ldots,o_{n-1})$ with each o_i a binary operation on $\mathbf{F}_i$, then the reduce operation $/o(\mathbf{a})$ is defined as

$$/o(\mathbf{a}) \equiv \mathbf{a}_1 o_1 (\mathbf{a}_2 o_2 (\ldots\ldots (\mathbf{a}_{n-2} o_{n-2} (\mathbf{a}_{n-1} o_{n-1} \mathbf{a}_n)) \ldots.).$$

Example 2.17.

Suppose $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_n)$ an $\mathbf{F}$ valued image and $\mathbf{F} = \mathbf{R}^n$, then

$$/+(\mathbf{a}) = \sum_{i=1}^{k} \mathbf{a}_i = \mathbf{a}_1 + \mathbf{a}_2 + \ldots. + \mathbf{a}_k$$

and

$$/\vee(\mathbf{a}) = \bigvee_{i=1}^{k} \mathbf{a}_i = \mathbf{a}_1 \vee \mathbf{a}_2 \vee \ldots. \vee \mathbf{a}_k$$

Observe that the vector valued image $\mathbf{a}$ was reduced to a real valued image.

Two elementary fusion/splitting functions of prime importance are the projection and injection functions. Let $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$. The projection onto the ith-coordinate is defined as before by

$$p_i : \mathbf{F} \rightarrow \mathbf{F}_i, \qquad (r_1, \ldots, r_n) \rightarrow r_i .$$

For $\mathbf{F}$ a homogeneous value set we define the injection function

$$q : \mathbf{F}_k \rightarrow \mathbf{F}, \text{ by } \; q(r) = (r, r, \ldots, r)$$

The ith-coordinate injection $q_i : \mathbf{F} \rightarrow \mathbf{F}$ is defined as $q_i \equiv q p_i$. Thus q_i replaces all the coordinate values of a point by the value of the ith-coordinate, namely

$$q_i(r_1, r_2, \ldots, r_i, \ldots, r_n) = (r_i, r_i, \ldots, r_i)$$

If $\mathbf{F} = \prod_{i=1}^{n} \mathbf{F}_i$ and $\mathbf{F}_j = \mathbf{R}$ for some j between 1 and n, then $\mathbf{F}^X$ forms a lattice. In particular, we define the jth-coordinate maximum and minimum of two $\mathbf{F}$ valued images as

$$\text{i.} \quad \mathbf{a} \vee|_j \mathbf{b} = \mathbf{a} * q_j[\chi_{\geq \mathbf{b}}(\mathbf{a})] + \mathbf{b} * q_j[\overline{\chi_{\geq \mathbf{b}}(\mathbf{a})}]$$

and

$$\text{ii.} \quad \mathbf{a} \wedge|_j \mathbf{b} = \mathbf{a} * q_j[\chi_{\leq \mathbf{b}}(\mathbf{a})] + \mathbf{b} * q_j[\overline{\chi_{\leq \mathbf{b}}(\mathbf{a})}],$$

respectively, where $\overline{[\chi_{\geq \mathbf{b}}(\mathbf{a})]}$ denotes the Boolean complement of $\chi_{\geq \mathbf{b}}(\mathbf{a})$. Thus,

$$\mathbf{a} \vee|_j \mathbf{b} = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) : \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) \text{ if } \mathbf{a}_j(\mathbf{x}) \geq \mathbf{b}_j(\mathbf{x}), \text{ otherwise } \mathbf{c}(\mathbf{x}) = \mathbf{b}(\mathbf{x}) \}.$$

Example 2.18. Directional Edge Detection

Directional edge detection is accomplished by convolving the image with the following six 3x3 edge masks with each mask having a direction associated with it.

$\mathbf{m}_1 =$

1	1	1
0	0	0
-1	-1	-1

180°

$\mathbf{m}_2 =$

-1	0	1
-1	0	1
-1	0	1

90°

$\mathbf{m}_3 =$

.7	1	1
-.8	0	.8
-1	-1	-.7

150°

$\mathbf{m}_4 =$

1	1	.7
.8	0	-.8
-.7	-1	-1

210°

$\mathbf{m}_5 =$

-.7	.8	1
-1	0	1
-1	-.8	.7

120°

$\mathbf{m}_6 =$

1	.8	-.7
1	0	-1
.7	-.8	-1

240°

Figure 13. The Six Directional Edge Detection Masks

The resulting six images are then fused to form a single 2-valued image **b**. Data fusion is accomplished by assigning to the resultant pixel the value of the largest magnitude of the corresponding pixels in the six images and either assigning the direction θ associated with the mask of the convolved image if the specific pixel value of the convolved image is positive, or $\theta + 180°$ mod 360° if the value is

negative. Thus, each pixel of **b** has both a magnitude and a direction associated with it. The Image Algebra translation of this algorithm is as follows. Let θ_i denote the direction in degrees associated with $\mathbf{m}_i$. Let f: $\mathbf{R} \rightarrow \mathbf{R}^2$ be defined by $f(r) = (|r|, 180\chi_{<0}(r))$, then

i. $\mathbf{a}_i = (0,\theta_i) + f(\mathbf{a} \oplus \mathbf{m}_i)$, i=1,...,6 and

ii. $\mathbf{b} = (\vee|_{1}) \overset{6}{\underset{i=1}{}} \mathbf{a}_i$

The next three figures provide a pictorial example of this algorithm.

Figure 14. Input Image **a**

Figure 15. Magnitude Image $p_1(\mathbf{b})$

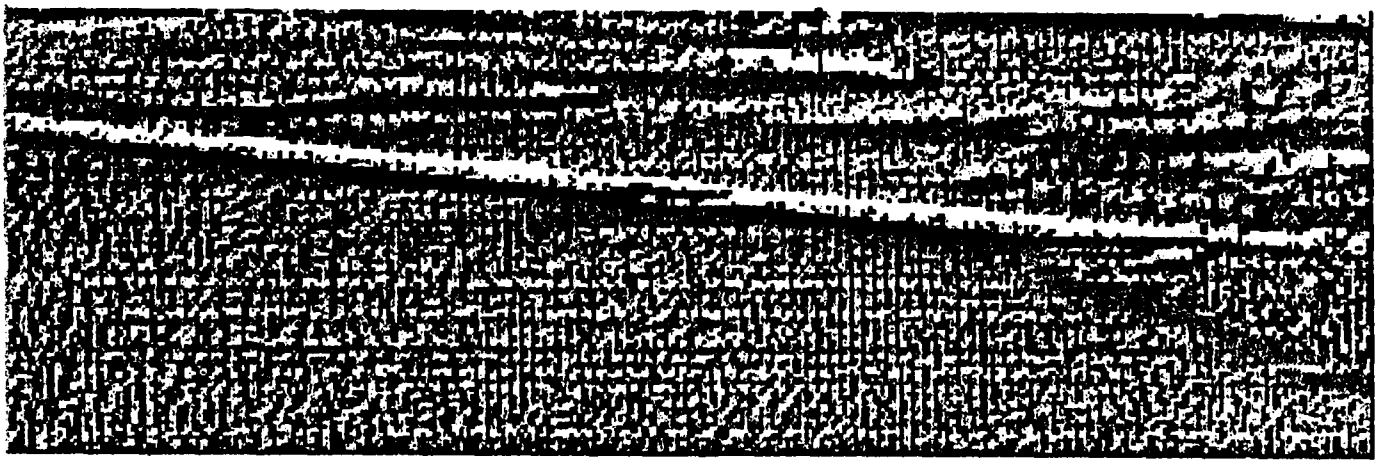


Figure 16. Degree Image $p_2(\mathbf{b})$

12. MULTILEVEL TEMPLATES AND MULTILEVEL TEMPLATE OPERATIONS

In this section we assume that $\mathbf{X}$ is of form $\mathbf{X}=\prod_{i=1}^{n}\mathbf{X}_i$ and $\mathbf{F}=\prod_{i=1}^{n}\mathbf{F}_i$. Let $Z\in\prod_{i=1}^{n}(2^{X_i})$, then Z is the form $Z=(Z_1,Z_2,.....,Z_n)$, where each $Z_i\subset\mathbf{X}_i$. We define a function

$$h:\prod_{i=1}^{n}(2^{X_i})\rightarrow 2^{X}=2^{(\prod_{i=1}^{n}X_i)}\text{ by}$$

$$h(Z) = Z_1 \times Z_2 \times \ldots\ldots \times Z_n$$

In an analogous fashion, let $(\mathbf{a}_1,\mathbf{a}_2,\ldots\ldots,\mathbf{a}_n) \in \prod_{i=1}^{n} (\mathbf{F}_i^{\mathbf{X}_i})$ and define $g: \prod_{i=1}^{n} (\mathbf{F}_i^{\mathbf{X}_i}) \rightarrow \mathbf{F}^{\mathbf{X}}$ by

$$g(\mathbf{a}_1,\mathbf{a}_2,\ldots\ldots,\mathbf{a}_n) = \mathbf{a} = \{(\mathbf{x},\mathbf{a}(\mathbf{x})) : \mathbf{a}(\mathbf{x}) = (\mathbf{a}_1(\mathbf{x}_1),\mathbf{a}_2(\mathbf{x}_2), \ldots, \mathbf{a}_n(\mathbf{x}_n))\}$$

Observe that g is a one-to-one and onto function from $\prod_{i=1}^{n} (\mathbf{F}_i^{\mathbf{X}_i})$ to $g(\prod_{i=1}^{n} (\mathbf{F}_i^{\mathbf{X}_i})) \subset \mathbf{F}^{\mathbf{X}}$; i.e., g embeds multisensor images into $\mathbf{F}^{\mathbf{X}}$. Similarly, h is a one-to-one and onto function from $\prod_{i=1}^{n} (2^{\mathbf{X}_i})$ to $h(\prod_{i=1}^{n} (2^{\mathbf{X}_i}))$. A template $\mathbf{t} \in \mathbf{F}_{\mathbf{Y} \rightarrow \mathbf{X}}$ is called a multilevel template if its range is

$$h(\prod_{i=1}^{n} (2^{\mathbf{X}_i})) \times g(\prod_{i=1}^{n} (\mathbf{F}_i^{\mathbf{X}_i})) \subset 2^{\mathbf{X}} \times \mathbf{F}^{\mathbf{X}}$$

Given a multilevel template $\mathbf{t} = (\mathcal{T}, t)$, we can define $\mathcal{T} : \mathbf{Y} \rightarrow 2^{\mathbf{X}_i}$ by $\mathcal{T}_i = p_i h^{-1} \mathcal{T}$, where p_i denotes the ith-coordinate projection $\prod_{i=1}^{n} (2^{\mathbf{X}_i}) \rightarrow 2^{\mathbf{X}_i}$ and, similarly, $t_i : \mathbf{Y} \rightarrow \mathbf{F}_i^{\mathbf{X}_i}$ by $t_i = p'_i g^{-1} t$, where p'_i denotes the ith-coordinate projection $\prod_{i=1}^{n} (\mathbf{F}_i^{\mathbf{X}_i}) \rightarrow \mathbf{F}_i^{\mathbf{X}_i}$. Thus, a multilevel template t can be thought of as a stack of templates $\mathbf{t} = (\mathbf{t}_1,\mathbf{t}_2,\ldots\ldots,\mathbf{t}_n)$, where each ith-coordinate $\mathbf{t}_i = (\mathcal{T}_i, t_i) \in (\mathbf{F}_i)_{\mathbf{Y} \rightarrow \mathbf{X}_i}$.

Operations between multivalued or multisensor images and multilevel templates are natural extensions of previously defined operations between images and templates. Suppose $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$ and $\mathbf{t} \in \mathbf{F}_{\mathbf{Y} \rightarrow \mathbf{W}}$ the multilevel template $\mathbf{t} = (\mathbf{t}_1,\mathbf{t}_2,\ldots\ldots,\mathbf{t}_n)$, where $\mathbf{W} = \prod_{i=1}^{n} \mathbf{W}_i$ and each pair $\mathbf{W}_i, \mathbf{X}_i$ being subsets of the same coordinate space. Let $o = (o_1,\ldots\ldots,o_n)$, where each o_i is an operation between $\mathbf{F}_i$ valued images on $\mathbf{X}_i$ and $\mathbf{F}_i$ valued templates from $\mathbf{Y}$ to $\mathbf{W}_i$. Then $\mathbf{a}\, o\, \mathbf{t}$ is defined as

$$\mathbf{a}\, o\, \mathbf{t} \equiv (\mathbf{a}_1\, o_1\, \mathbf{t}_1,\ldots\ldots,\mathbf{a}_n\, o_n\, \mathbf{t}_n).$$

Example 2.19.

Suppose $\mathbf{F} = \mathbf{R}^3$ and $o = (\oplus, \boxtimes, \otimes)$, then

$$\mathbf{a}\, o\, \mathbf{t} \equiv (\mathbf{a}_1 \oplus \mathbf{t}_1, \mathbf{a}_2 \boxtimes \mathbf{t}_2, \mathbf{a}_1 \otimes \mathbf{t}_3)$$

If o is homogeneous, say $o = (\oplus, \oplus, \oplus)$, then - following our earlier convention - we set $o = \oplus$ and define

$$\mathbf{a} \oplus \mathbf{t} \equiv (\mathbf{a}_1 \oplus \mathbf{t}_1, \mathbf{a}_2 \oplus \mathbf{t}_2, \mathbf{a}_1 \oplus \mathbf{t}_3)$$

It follows from the above discussion that a multilevel template can have different configurations at different levels $\mathbf{X}_i$ and can operate differently on the different levels of a multivalue or multisensor image.

SECTION IV
RELATION OF IMAGE ALGEBRA TO OTHER MATHEMATICAL STRUCTURES

One of the main tools of algebra is the isomorphism. An algebraic structure can be thought of as a set of symbols and rules for combining the symbols. The structure can be used to model systems which obey those rules. If two algebraic structures are isomorphic, that is, if there exists a 1-1, operation preserving mapping of one onto the other, then they provide two different viewpoints of the same situation. The idea is that the more ways one has to look at a problem the better chance there is of solving it.

Developing an algebraic structure for digital image processing enables one to take advantage of isomorphisms. In this chapter we establish isomorphisms between the image algebra and the mathematical structures known as linear algebra, polynomial algebra, and lattice algebras. We also show how these relationships can be applied to the problem of developing systematic techniques for the optimization and derivation of parallel algorithms.

1. RELATIONSHIP BETWEEN IMAGE ALGEBRA AND LINEAR ALGEBRA

A subalgebra of the image algebra is a subcollection of the set of operators and operands of the image algebra. In this section a precise relationship between a subalgebra of the image algebra and linear algebra will be described which shows how all the operators and operands of finite dimensional linear algebra (matrices, vectors, scalars and the operations between them) are modeled by the image algebra. One consequence of the relationship is that any linear image-to-image transform can be written as $\mathbf{a} \oplus \mathbf{k}$ for some template $\mathbf{k}$. It will then be shown how the above relationship, along with the techniques of template decomposition and configuration restriction, can be used to develop algorithms for implementing linear transforms of images on arrays of processors with various interconnection schemes.

Let $\mathbf{X}$ be an arbitrary bounded subset of $\mathbf{Z}^k$ and $\mathbf{F}$ a field. Observe that if $\mathbf{s}, \mathbf{t} \in \mathbf{F}_{\mathbf{Y}\to\mathbf{X}}$ such that $s_y(\mathbf{x}) = t_y(\mathbf{x})$ for every $\mathbf{x} \in \mathbf{X}$ and $\mathbf{y} \in \mathbf{Y}$, then $\mathbf{a} \oplus \mathbf{t} = \mathbf{a} \oplus \mathbf{s}$ for every $\mathbf{a} \in \mathbf{F}_{\mathbf{Y}\to\mathbf{X}}$. Thus, padding a configuration with zeros has no effect on the computation of $\mathbf{a} \oplus \mathbf{t}$. We will call two templates equivalent if the above is true. This definition of equivalence imposes an equivalence relation on $\mathbf{F}_{\mathbf{Y}\to\mathbf{X}}$. The collection of equivalence classes forms a ring (called a quotient ring of $\mathbf{F}_{\mathbf{Y}\to\mathbf{X}}$) which we will denote by $L_{\mathbf{Y}\to\mathbf{X}}$. We will call $L_{\mathbf{Y}\to\mathbf{X}}$ the ring of templates with minimal configuration. Practically speaking, $L_{\mathbf{Y}\to\mathbf{X}} = \{ \mathbf{t} \in \mathbf{F}_{\mathbf{Y}\to\mathbf{X}} : t_y(\mathbf{x}) \neq 0$ if

and only if $\mathbf{x} \in \mathcal{T}(\mathbf{y})$ } and if two elements of $L_{Y\to X}$ are combined using + or $\oplus$ the result is again in $L_{Y\to X}$, i.e. has minimal configuration. Denote by **0** the template in $L_{Y\to X}$ with empty configuration at every point. Then $\mathbf{0}\oplus\mathbf{t} = \mathbf{t}\oplus\mathbf{0} = \mathbf{0}$ for every template $\mathbf{t} \in L_{Y\to X}$. Furthermore, for $\mathbf{s}, \mathbf{t} \in L_{Y\to X}$, $\mathbf{t} - \mathbf{s} = \mathbf{0}$ if and only if $\mathbf{t} = \mathbf{s}$ (This property does not hold for $F_{Y\to X}$). Whenever we consider $\oplus$ we will assume that the template has minimal configuration.

Since $\mathbf{X}$ is a finite set it can be linearly ordered. Thus we can write $\mathbf{X} = \{x_0, x_1, \ldots, x_{n-1}\}$. We define a mapping $\nu : \mathbf{F}^{\mathbf{X}} \to \mathbf{F}^{\mathbf{n}}$ by $\nu(\mathbf{a}) = (a(x_0), a(x_1), \ldots, a(x_{n-1}))^t$. Since $\nu(r\mathbf{a}+s\mathbf{b}) = r\nu(\mathbf{a})+s\nu(\mathbf{b})$ and ν is 1-1 and onto, ν is a vector space isomorphism.

Let $L_X \equiv L_{X\to X}$ and $(M_n, *, +)$ denote the ring of n x n matrices with entries from $\mathbf{F}$ under matrix multiplication and addition. For any $\mathbf{t} \in L_X$ we define a matrix $M_t = (m_{ij})$ where $m_{ij} \equiv t_{x_i}(x_j)$. Note that the i^{th} row of M_t is $\nu(t_{x_i})^t$. Define a mapping $\Psi : L_X \to M_n$ by $\Psi(\mathbf{t}) = M_t$.

Theorem 4.1.1. Ψ is a ring isomorphism of $(L_X, \oplus, +)$ onto $(M_n, *, +)$. That is, if $\mathbf{t}, \mathbf{s} \in L_X$ then

i.) $\Psi(\mathbf{s}+\mathbf{t}) = \Psi(\mathbf{s})+\Psi(\mathbf{t})$ or $M_{s+t} = M_s + M_t$,

ii.) $\Psi(\mathbf{s}\oplus\mathbf{t}) = \Psi(\mathbf{t})\Psi(\mathbf{s})$ or $M_{s\oplus t} = M_t M_s$.

Proof: 1.) Let $\mathbf{t}, \mathbf{s} \in L_X$. Then

$$\Psi(\mathbf{s}+\mathbf{t}) = \begin{bmatrix} \nu((s+t)_{x_0})^t \\ \vdots \\ \nu((s+t)_{x_{n-1}})^t \end{bmatrix}$$

Since $\nu((s+t)_{x_i}) = \nu(s_{x_i} + t_{x_i}) = \nu(s_{x_i}) + \nu(t_{x_i})$, we may conclude that $\Psi(\mathbf{s}+\mathbf{t}) = \Psi(\mathbf{s})+\Psi(\mathbf{t})$.

2.) Let $\mathbf{r} = \mathbf{s}\oplus\mathbf{t}$. Then, by definition of $\oplus$, $r_{x_i}(x_j) = \sum_{x\in\mathcal{T}(x_i)} t_{x_i}(x) s_x(x_j)$. By definition of

matrix multiplication $M_t M_s = (c_{ij})$ where $c_{ij} = \sum_{k=0}^{n-1} t_{x_i}(x_k) s_{x_k}(x_j)$. But since $t_{x_i}(x_k) = 0$ if $x_k \notin T(x_i)$ we must have $c_{ij} = r_{x_i}(x_j)$ which implies that $\Psi(s \oplus t) = \Psi(t)\Psi(s)$.

Ψ is clearly onto. Furthermore Ψ is 1-1 since if $\Psi(t) = \Psi(s)$ then $\Psi(s - t) = 0$. But 0 is the only template with all zero grey values and minimal configuration. Hence $(s - t) = 0$ so $s = t$.

Q.E.D.

Observe that if D is any n x n diagonal matrix then $t = \Psi^{-1}(D)$ has the property that if $b = \sum_{x \in X} t_x$ then for every $a \in F^X$, $a * b = a \oplus t$. Hence diagonal matrices correspond to templates which are essentially images, that is, the image b can be used to implement the transform $a \mapsto a \oplus t$.

We now prove a final theorem concerning the relationship between image and linear algebra.

Theorem 4.1.2. For every $a \in F^X$ and $t \in L_X$, $a \oplus t = \nu^{-1}(\Psi(t)\nu(a))$.

Proof: Let $b = a \oplus t$. Then, by definition of $\oplus$, $b(y) = \sum_{x \in T(y)} a(x) t_y(x)$. $\Psi(t)\nu(a) = (c_0, c_1, \ldots, c_{n-1})^t$ where $c_i = \sum_{k=0}^{n-1} t_{x_i}(x_k) a(x_k)$. Since $t_{x_i}(x_k) = 0$ if $x_k \notin T(x_i)$ we have that $c_i = b(x_i)$ which shows that $\nu(b) = \Psi(t)\nu(a)$. The conclusion follows from the fact that ν is invertible.

Q.E.D.

The above theorems have many consequences. Two important ones are that template composition corresponds to matrix multiplication and that the action of a template on an image via the $\oplus$ operation corresponds to the action of a matrix on a vector. Thus a subalgebra of the image algebra serves as an alternative algebraic formulation of finite dimensional linear operations. By incorporating the notion of configuration in the definition of templates a geometric component has been associated with a linear computation. This concept becomes important when trying to map algorithms to special architectures with certain geometric configurations associated with them. Since these theorems are true for any bounded $X \subset Z^k$ all linear transforms defined on any of the examples of the previous section

can be expressed using $\oplus$. Moreover, all decompositions of linear transforms can be expressed also.

The special case that $\mathbf{X}$ is an m x n array is of particular interest. We take $\mathbf{X} = \{ (x,y) : 0 \leq x \leq m\text{-}1, 0 \leq y \leq n\text{-}1 \}$. We can order $\mathbf{X}$ lexicographically in row major form by mapping $(x,y) \longmapsto xn+y$. The mappings ν and Ψ then have the form: $\nu(\mathbf{a}) = (\mathbf{a}_0,\mathbf{a}_1,\ldots,\mathbf{a}_{nm-1})^t$ where $\mathbf{a}_i = \mathbf{a}(x,y)$ if $i=xn+y$ and

$$\Psi(\mathbf{t}) = \begin{bmatrix} \nu(\mathbf{t}_{(0,0)})^t \\ \nu(\mathbf{t}_{(0,1)})^t \\ \vdots \\ \nu(\mathbf{t}_{(m-1,n-1)})^t \end{bmatrix}.$$

Henceforth, whenever $\mathbf{X}$ is an m x n array we shall assume that these particular mappings have been used.

2. SIMULATION OF ARRAYS OF PROCESSORS

The image algebra can be used to simulate an array of processors with limited communication by imposing restrictions on the template configurations allowed in translating algorithms. In fact, suppose an array of processors is modeled as a graph $G = (V(G),E(G))$ where the vertices represent the processors and the edges represent a communications link between processors. Suppose G is embedded in n-dimensional space. Then let $\mathbf{X} = V(G)$ and define a template $\mathbf{r} = \mathbf{r}(G) \in \mathbf{F}_{\mathbf{X}\to\mathbf{X}}$ by $\mathcal{R}(\mathbf{y}) = \{ \mathbf{x} : [\mathbf{y},\mathbf{x}] \in E(G) \}$ and $\mathbf{r}_\mathbf{y} = \{(\mathbf{x},\mathbf{r}_\mathbf{y}(\mathbf{x})) : \mathbf{r}_\mathbf{y}(\mathbf{x})=1$ for every $\mathbf{x} \in \mathcal{R}(\mathbf{x})\}$. We say that an expression $p(\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n)$ is compatible with the template restriction induced by $\mathbf{r}$ if for every template t appearing in p we have that $T(\mathbf{y}) \subset \mathcal{R}(\mathbf{y})$ for every $\mathbf{y} \in \mathbf{X}$. In other words, if p is compatible with $\mathbf{r}$ then the algorithm defined by p can, in principle, be implemented on the processor array modeled by G. In this way the image algebra can serve as an algebraic model for special architectures.

For example, assume that $\mathbf{X}$ is an n x m array and define the elementary von Neumann template $\mathbf{v}$ by $V(x,y)=\{(x+1,y),(x-1,y),(x,y+1),(x,y-1),(x,y)\}$ and $v_{(x,y)}(u,v) = 1$ if and only if $(u,v) \in V(x,y)$. Then, using $\mathbf{v}$ as a restriction template, the image algebra simulates a mesh connected array of processors. Thus, if $p(\mathbf{a}_1,\mathbf{a}_2) = \mathbf{a}_1 \oplus \mathbf{t}_1 - 2(\mathbf{a}_2 \boxtimes \mathbf{t}_2) + \mathbf{b} * (\mathbf{a}_1 * \mathbf{a}_2 \circledast \mathbf{t}_1)$ and $T_1(x,y), T_2(x,y) \subset V(x,y)$ for all $(x,y) \in \mathbf{X}$ then, in principle, the algorithm defined by $p(\mathbf{a}_1,\mathbf{a}_2)$

could be implemented on such a machine.

Another example is the hierarchical or pyramidal arrays. Let n be any positive integer and for i=0,1,...,n let $\mathbf{X}_i = \{ (x,y,z) : 1 \leq x,y \leq 3^{n-i}, z = i \}$. and $\mathbf{X} = \bigcup_{i=0}^{n} \mathbf{X}_i$. Define the restriction template $\mathbf{r} \in \mathbf{F}_{\mathbf{X} \to \mathbf{X}}$ by $\mathcal{R}(x,y,z) = \{ (u,v,w) :$

$w = z$ and (u,v) is an 8-neighbor of (x,y); or

$w = z\text{-}1, x = \left\lceil \frac{u}{3} \right\rceil$, and $y = \left\lceil \frac{v}{3} \right\rceil$; or

$w = z+1$, $u = \left\lceil \frac{x}{3} \right\rceil$, and $v = \left\lceil \frac{y}{3} \right\rceil \} \cap \mathbf{X}$.

The notation $\lceil a \rceil$ denotes the smallest integer greater than or equal to a. An image algebra expression p over $\mathbf{X}$ will be compatible with this pyramidal structure if for every template t appearing in p we have $\mathcal{T}(x,y,z) \subset \mathcal{R}(x,y,z)$ for every $(x,y,z) \in \mathbf{X}$. Thus, if $n = 2$ then the configuration of a template at the point (1,1,1) must consist of a subset of the following points:

the 8-neighbors in the same plane, (1,2,1), (2,2,1), (2,1,1);

the children, (3,3,0), (3,2,0), (3,1,0), (2,3,0), (2,2,0), (2,1,0), (1,3,0), (1,2,0), (1,1,0); and

the parent, (1,1,2).

Any image algebra expression over $\mathbf{X}$ involving only templates satisfying the template restriction induced by $\mathbf{r}$ defines an algorithm which could, in principle, be implemented on such a machine.

As a final example consider the binary n-cube, Reference 14. It can be represented as the set of n-tuples with coordinates either 0 or 1, that is, $\mathbf{X} = \{ (x_1,x_2,..., x_n) : x_i \in \{0,1\} \}$. Define the restriction template $\mathbf{r}$ by declaring $\mathbf{r}$ to be the elementary template with configuration function given by $\mathcal{R}(\mathbf{y}) = \{ \mathbf{x} : H(\mathbf{y},\mathbf{x}) \leq 1 \}$ where H denotes the Hamming distance.

These examples show how the image algebra can be used to simulate special architectures. Since the interconnection scheme of any processor array can be represented by a graph, virtually any processor array can be represented by the image algebra. Moreover, by our completeness theorems, References 12 and 10, not only can the arrays themselves be represented but the transformations defined on the arrays can be represented also.

3. TEMPLATE DECOMPOSITIONS AND MATRIX FACTORIZATIONS

We can combine the theorems on the relationships between image and linear algebra with the use of the image algebra as an algebraic model for various processor arrays to mathematically formulate the problem of mapping linear transforms to specific hardware. If a processor array is modeled as a graph G, $\mathbf{r}$ is the restriction template associated with G and $\mathbf{t}$ is a template compatible with $\mathbf{r}$, then we say that $\mathbf{t}$ and M_t are local with respect to $\mathbf{r}$.

A template restriction will result in a particular matrix structure. For example, the von Neumann restriction will impose the condition that a matrix corresponding to an allowable template will have a certain banded structure. Many techniques for decomposing matrices have been developed. If $\mathbf{t}$ is a template and if a local decomposition of M_t can be found, then the mapping Ψ^{-1} can be used to construct a local decomposition of $\mathbf{t}$. Moreover, because of the algebraic relations of associativity and distributivity, the local decompositions will result in local algorithms. That is, if $M_t = M_1 M_2$, $\mathbf{t}_1 = \Psi^{-1}(M_1)$, and $\mathbf{t}_2 = \Psi^{-1}(M_2)$, then $\Psi^{-1}(M_t) = \mathbf{t} = \mathbf{t}_2 \oplus \mathbf{t}_1$ and $\mathbf{a} \oplus \mathbf{t} = \mathbf{a} \oplus (\mathbf{t}_2 \oplus \mathbf{t}_1) = (\mathbf{a} \oplus \mathbf{t}_2) \oplus \mathbf{t}_1$. Also if $M_t = M_1 + M_2$, then $\Psi^{-1}(M_t) = \mathbf{t} = \mathbf{t}_1 + \mathbf{t}_2$ and $\mathbf{a} \oplus \mathbf{t} = \mathbf{a} \oplus (\mathbf{t}_1 + \mathbf{t}_2) = \mathbf{a} \oplus \mathbf{t}_1 + \mathbf{a} \oplus \mathbf{t}_2$. Hence we can decompose, or factor, a template $\mathbf{t}$ into sums and products of local templates by decomposing, or factoring, the corresponding matrix.

Observe that factoring into products implies that the result of a computation can be operated on at the next step whereas decomposing into sums implies that two images must be computed separately, which requires twice as much storage space. Since all the operations are preserved under the mapping Ψ^{-1} such a decomposition will result in an algorithm for the local computation of $\mathbf{a} \oplus \mathbf{t}$. In this way local algorithms can be developed using the interaction between the image algebra and linear algebra. The algorithms can be thought of as being mapped onto a specific hardware using Ψ^{-1}. Therefore, we can use the image algebra to mathematically formulate the problem of mapping linear algorithms to specific hardware as follows:

Given a template $\mathbf{t}$, find a decomposition of $\mathbf{t}$ in terms of sums and products of local templates, or, equivalently, given a matrix M, find a decomposition of M in terms of sums and products of local matrices.

We have used these ideas to develop methods for deriving algorithms for the computation of two-dimensional discrete Fourier transforms of arbitrary size on mesh connected arrays. We show how to decompose the two-dimensional Fourier template into products of

local templates. These methods will be discussed in the subsequent sections.

4. IMAGE ALGEBRA AND POLYNOMIAL ALGEBRA

In this section precise relationships between polynomial algebra and the image algebra are described. It is shown how these relationships can be used to develop systematic methods for designing parallel algorithms for circulant image transforms. The methodology is similar to that of the previous section in that we seek to decompose templates by decomposing other algebraic objects.

Throughout this section it will be assumed that $\mathbf{X}$ is an m x n array. Everything that will be done generalizes easily to the case that $\mathbf{X}$ is a rectangular parallelepiped but the notation becomes cumbersome. We will show that the ring of circulant templates on $\mathbf{X}$ is isomorphic to a quotient of the ring of polynomials in two variables. We exhibit a class of mappings which are related to this isomorphism and which are almost isomorphisms themselves.

We begin with some preliminary definitions and notation. If $\mathbf{x} \in \mathbf{Z}^k$ then $\mathbf{x}$ $\text{mod}(n_1,n_2,\ldots, n_k) \equiv (x_1 \text{ mod} n_1, x_2 \text{ mod} n_2, \ldots, x_k \text{ mod} n_k)$.

Definition 4.4.1. If $\phi : \mathbf{X} \rightarrow \mathbf{X}$ is of the form $\phi(\mathbf{x}) = (\mathbf{x}+\mathbf{z}) \text{ mod}(m,n)$ Then ϕ is called a circulant translation.

Definition 4.4.2. If $\mathbf{t} \in \mathbf{F}_{\mathbf{X}\rightarrow\mathbf{X}}$ satisfies the following two properties for every circulant translation ϕ:

$\phi T(\mathbf{y}) = T(\phi(\mathbf{y}))$ for every $\mathbf{y} \in \mathbf{X}$ and

$t_{\phi(\mathbf{y})}(\phi(\mathbf{x})) = t_{\mathbf{y}}(\mathbf{x})$ for every $\mathbf{x},\mathbf{y} \in \mathbf{X}$,

then $\mathbf{t}$ is called a circulant template.

A circulant template wraps around at the boundary. We denote the ring, under the operations $\oplus$ and $+$, of all circulant templates in L_X by C_X. If $\mathbf{t}$ is a circulant template then we call a mapping $f : \mathbf{F}^{\mathbf{X}} \rightarrow \mathbf{F}^{\mathbf{X}}$ of the form $f(\mathbf{a}) = \mathbf{a} \oplus \mathbf{t}$ a circulant transform. Observe that $\mathbf{a} \oplus \mathbf{t}$ denotes the circular convolution of the images $\mathbf{a}$ and $\mathbf{b}$ where $\mathbf{b}(i,j) = t_{(0,0)}((-i,-j) \text{ mod}(m,n))$. Thus, circulant templates are used to implement circular convolutions which are standard techniques in digital signal and image processing.

Let $K[x,y]/(x^m-1,y^n-1)$ denote the ring of polynomials in two variables where addition is performed in the usual way and multiplication is performed as usual except that the substitutions $x^m = 1$ and $y^n = 1$ can be made at any time. We say that a polynomial is in reduced form in this ring if all possible substitutions of that type have been made. Two polynomials

in this ring are declared to be equivalent if they have the same reduced form. For example let $m = 3$ and $n = 2$. Then $x^3 = 1$ and $y^2 = 1$. Let $f(x,y) = 1+x+yx^2$ and $g(x,y) = 1+yx+x^2$. Then $f(x,y) + g(x,y) = 2+x+yx+yx^2+x^2 = 2+(1+y)x+(1+y)x^2$. Also $f(x,y)g(x,y) = 1+yx+x+x^2+x+yx^2+x^3+yx^2+y^2x^3+yx^4$. The reduced form of $f(x,y)g(x,y)$ is $1+yx+x^2+x+yx^2+1+yx^2+1+yx = 3+(2y+1)x+(2y+1)x^2$.

For every $\mathbf{z} \in \mathbf{X}$ we define a mapping $\Gamma_z : C_X \rightarrow K[x,y]/(x^m-1,y^n-1)$ by

$$\Gamma_z(\mathbf{t}) \equiv p_t(x,y,z) \equiv \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} t_z(i,j)x^iy^j.$$

If $\mathbf{t}$ is a circulant template then Γ_z is called a polynomial representative of $\mathbf{t}$. It is easy to see that a circulant template $\mathbf{t}$ is local with respect to a mesh connected array if $p_t(x,y,0) = x\text{-}a$ or $y\text{-}a$ for some a. We also point out that if the polynomial corresponding to a circulant template $\mathbf{t}$ is of the form x^iy^j then the circulant transform $\mathbf{a} \mapsto \mathbf{a} \oplus \mathbf{t}$ simply circularly shifts all the grey levels i units vertically and j units horizontally.

Theorem 4.4.3. If $\mathbf{s}$ and $\mathbf{t}$ are circulant templates, then $\Gamma_0(\mathbf{s} \oplus \mathbf{t}) = \Gamma_0(\mathbf{s})\Gamma_0(\mathbf{t})$.

Proof: For any $\mathbf{z} \in \mathbf{X}$ denote by ϕ_z the circulant translation defined by $\phi_z(\mathbf{x}) = (\mathbf{x}\text{-}\mathbf{z})$ mod(m,n). Note that $\phi_z(\mathbf{z}) = \mathbf{0}$. Recall that, if $\mathbf{r} = \mathbf{s} \oplus \mathbf{t}$, then

$$\mathbf{r}_0 = \{ (\mathbf{y}, \mathbf{r}_0(\mathbf{y})) : \mathbf{r}_0(\mathbf{y}) = \sum_{\mathbf{z} \in T(0)} \mathbf{t}_0(\mathbf{z})\mathbf{s}_z(\mathbf{y}) \}.$$

Since $\mathbf{s}$ is a circulant $\mathbf{s}_z(\mathbf{y}) = \mathbf{s}_0(\phi_z(\mathbf{y})) = \mathbf{s}_0((\mathbf{y}\text{-}\mathbf{z}) \bmod(m,n))$. But by definition of polynomial multiplication (with $x^m \equiv y^n \equiv 1$) the numbers $\sum_{\mathbf{z} \in T(0)} \mathbf{t}_0(\mathbf{z})\mathbf{s}_0((\mathbf{y}-\mathbf{z}) \bmod(m,n))$ are precisely the coefficients of the polynomial product $p_s(x,y,0)p_t(x,y,0)$.

Q.E.D.

Theorem 4.4.4. For every $\mathbf{z} = (j,k) \in \mathbf{X}$, Γ_z is 1-1 and onto. Moreover, if $\mathbf{s}, \mathbf{t} \in C_X$, then

$$\Gamma_z(\mathbf{s})+\Gamma_z(\mathbf{t}) = \Gamma_z(\mathbf{s}+\mathbf{t}), \tag{7}$$

$$\Gamma_z(\mathbf{t}) = x^j y^k \Gamma_0(\mathbf{t}), \text{ and} \tag{8}$$

$$\Gamma_z(\mathbf{s} \oplus \mathbf{t}) x^j y^k = \Gamma_z(\mathbf{s}) \Gamma_z(\mathbf{t}). \tag{9}$$

Proof: Γ_z is clearly onto. Γ_z is 1-1 since C_X contains only circulant templates with minimal configuration. Equation (7) follows from the fact that addition of templates is defined pointwise.

To prove (8) let ϕ be the circulant translation defined by $\phi(\mathbf{x}) = (\mathbf{x}+\mathbf{z}) \bmod(m,n)$. Note that $\phi(\mathbf{0}) = \mathbf{z}$. Let $\mathbf{t}$ be an arbitrary circulant template. Then

$$x^j y^k p_t(x,y,0) = x^j x^k \sum_{i=0}^{m-1} \sum_{h=0}^{n-1} t_0(i,h) x^i y^h =$$

$$= \sum_{i=0}^{m-1} \sum_{h=0}^{n-1} t_z((i+j) \bmod m,(h+k) \bmod n) x^{i+j} y^{h+k} =$$

$$= \sum_{i=0}^{m-1} \sum_{h=0}^{n-1} t_z(i,h) x^i y^h = p_t(x,y,\mathbf{z}).$$

since $x^m \equiv y^n \equiv 1$ and $\mathbf{t}$ is circulant. This enables us to deduce that $\Gamma_z(\mathbf{t}) = x^j y^k \Gamma_0(\mathbf{t})$ or $x^{m-j} y^{n-k} \Gamma_z(\mathbf{t}) = \Gamma_0(\mathbf{t})$.

To prove (9) observe that since $\Gamma_0(\mathbf{s} \oplus \mathbf{t}) = \Gamma_0(\mathbf{s}) \Gamma_0(\mathbf{t})$ we have that

$$\Gamma_z(\mathbf{s} \oplus \mathbf{t}) = x^j y^k \Gamma_0(\mathbf{s}) \Gamma_0(\mathbf{t}) = x^j y^k \left[x^{m-j} y^{n-k} \Gamma_z(\mathbf{s}) x^{m-j} y^{n-k} \Gamma_z(\mathbf{t}) \right] = x^{m-j} y^{n-k} \Gamma_z(\mathbf{s}) \Gamma_z(\mathbf{t}).$$

The desired result now follows by multiplying the last equation by $x^j y^k$.

Q.E.D.

Corollary 4.4.5. Γ_0 is an isomorphism.

Corollary 4.4.6. Let $\mathbf{z}_1 = (i,j)$, $\mathbf{z}_2 = (s,t) \in \mathbf{X}$ and $\mathbf{t} \in C_X$. If $p_t(x,y,\mathbf{z}_1) = p_1(x,y) p_2(x,y)$, then $p_t(x,y,\mathbf{z}_2) = q_1(x,y) q_2(x,y)$ where $q_1(x,y) = x^{s-i} y^{t-j} p_1(x,y)$ and $q_2(x,y) = p_2(x,y)$.

Corollary 4.4.7. If $\mathbf{z} = (i,j) \in \mathbf{X}$, $\mathbf{t} \in C_{\mathbf{X}}$, and $p_t(x,y,\mathbf{z}) = p_1(x,y)p_2(x,y)$, then $\mathbf{t} = \mathbf{t_1} \oplus \mathbf{t_2} \oplus \mathbf{s}$ where $\mathbf{s}$ represents a circular shift of n-j units horizontally and m-i units vertically. Moreover, since $\mathbf{s}$ represents a circular shift it can be replaced by a template $\hat{\mathbf{s}}$ which represents a circular shift j units horizontally and i vertically.

The $\Gamma_{\mathbf{z}}$'s constitute the class of mappings between image and polynomial algebra which are almost isomorphisms. Since they differ from isomorphisms only by shifts they can be used in the same way. Clearly if any of the polynomial representatives of a template can be factored the template can be factored correspondingly. We now examine some specific consequences of these theorems. We will use them to show how any separable circulant template can be computed locally with respect to the von Neumann configuration and give upper bounds on the number of parallel steps required to do so. Since the von Neumann restriction on an m x n array simulates mesh connected arrays these methods can be used on such machines.

a. Parallel Algorithms via Polynomial Factorization

Definition 4.4.8. If $f(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} x^i y^j$ then

$\deg(f(x,y)) = \max \{ i+j : a_{ij} \neq 0 \}$.

$\deg_x(f(x,y)) = \max \{ i : a_{ij} \neq 0 \}$.

$\deg_y(f(x,y)) = \max \{ j : a_{ij} \neq 0 \}$.

Definition 4.4.9. If $\mathbf{t}$ is a circulant template and $\deg(p_t(x,y,\mathbf{z})) \leq \deg(p_t(x,y,\mathbf{w}))$ for every $\mathbf{w} \in \mathbf{X}$ then we say that $\mathbf{z}$ is a minimal point for $\mathbf{t}$.

Definition 4.4.10. Let $\mathbf{t}$ be a circulant template. We say that $\mathbf{t}$ is separable if there exists polynomials f(x) and g(y) such that $p_t(x,y,0) = f(x)g(y)$. By the preceding corollaries if $\mathbf{t}$ is separable then for every $\mathbf{z} \in \mathbf{X}$ there exists polynomials f(x) and g(y) such that $p_t(x,y,\mathbf{z}) = f(::)g(y)$.

We will make use of the fundamental theorem of algebra. We state it here for reference.

Fundamental Theorem of Algebra: If f(x) is a polynomial in one variable with complex coefficients then f(x) can be factored into n linear factors. That is, $f(x) = a(x-r_1)(x-r_2)\cdots(x-r_n)$ where $a, r_1, r_2, \cdots, r_n$ are (possibly) complex numbers.

The following theorem is a very satisfying result which yields a systematic method for computing any separable circulant transform locally with respect to a 4-connected processor array and gives an upper bound on the number of parallel steps required to do so.

Theorem 4.4.11. Let $\mathbf{t}$ be a separable, circulant template and let $\mathbf{z} = (s,t)$ be a minimal point for $\mathbf{t}$. Let $k = \deg_x(p_t(x,y,\mathbf{z}))$ and $j = \deg_y(p_t(x,y,\mathbf{z}))$. Let $v = \min(s, m-s)$ and $h = \min(t, n-t)$. Then the circulant transform $\mathbf{a} \rightarrow \mathbf{a} \oplus \mathbf{t}$ can be computed in at most v+h+k+j+1 local, parallel steps. v+h of these steps consist of vertical or horizontal circular shifts of the entire array by one location, k+j of the steps consist of at most one addition and one multiplication (possibly complex) per pixel, and one of the steps consists of at most one multiplication per pixel.

Proof: Since $\mathbf{t}$ is separable there exists f(x) and g(y) such that $p_t(x,y,\mathbf{z}) = f(x)g(y)$. By assumption $\deg(f(x)) = k$ and $\deg(g(y)) = j$. By the fundamental theorem of algebra $f(x) = a(x-q_1)(x-q_2)\ldots(x-q_k)$ and $g(y) = b(y-r_1)(y-r_2)\ldots(y-r_j)$ where $a, b, q_1, \ldots, q_k, r_1, \ldots, r_j$ are (possibly) complex numbers. Choose circulant templates $\mathbf{q}$, $\mathbf{r}$, $\mathbf{q}_1, \ldots, \mathbf{q}_k$, $\mathbf{r}_1, \ldots, \mathbf{r}_j$ such that $\Gamma_0(\mathbf{q}) = f(x)$, $\Gamma_0(\mathbf{r}) = g(y)$, $\Gamma_0(\mathbf{q}_i) = x - q_i$, and $\Gamma_0(\mathbf{r}_i) = y - r_i$. Then $\Gamma_z(\mathbf{t}) = \Gamma_0(\mathbf{q})\Gamma_0(\mathbf{r})$. By Corollary 4.4.7, $\mathbf{t} = \mathbf{q} \oplus \mathbf{r} \oplus \mathbf{s}$ where $\mathbf{s}$ represents a shift of v locations in the vertical direction and h in the horizontal direction. Now by Theorem 4.4.3, $\Gamma_0(\mathbf{q}) = a(\prod_{i=1}^{k} \Gamma_0(\mathbf{q}_i))$ implies that $\mathbf{q} = a(\bigoplus_{i=1}^{k} \mathbf{q}_i)$. Similarly $\mathbf{r} = b(\bigoplus_{i=1}^{j} \mathbf{r}_i)$. Hence $\mathbf{t} = ab\left[(\bigoplus_{i=1}^{k} \mathbf{q}_i) \oplus (\bigoplus_{i=1}^{j} \mathbf{r}_i) \oplus \mathbf{s}\right]$. Since templates corresponding to linear polynomials in one variable are local and shifts are obviously local this last equation expresses $\mathbf{t}$ as a product of local templates. This concludes the proof.

Q.E.D.

The templates are shown in Figure 17. In this case, one thinks of the templates as masks. Thus, convolving an input image with the masks $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k$ will result in the same output image as convolving the input image with the k x 1 mask $\mathbf{q} = \bigoplus_{i=1}^{k} \mathbf{q}_i$.

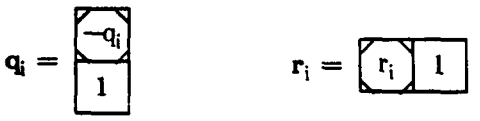


Figure 17. Templates Required to Compute a Separable Circulant Transform Locally

As an example suppose $\mathbf{X}$ is a 512 x 512 array and $\mathbf{t}$ is a 30 x 30 separable template. Then $\mathbf{z} = (14,14)$ and $v = h = 14$. Hence, if $\mathbf{a} \in \mathbf{F}^{\mathbf{X}}$, then $\mathbf{a} \rightarrow \mathbf{a} \oplus \mathbf{t}$ can be computed locally in parallel with 60 parallel steps consisting of at most one multiplication and addition per pixel, 1 step consisting of one multiplication per point, and a total of 28 unit horizontal or vertical circular shifts. Since the computation in its original form required 900 multiplications, it is clear that template decompositions can be used to derive algorithms which are more efficient with respect to the number of arithmetic operations as well as parallel.

Nontrivial examples of such templates are the discretizations of the Marr-Hildreth edge operators, Reference 15. Other examples include a class of templates obtained by computing the inverses of blurring transformations, a specific example of which will be presented in the next section.

If a circulant template is not separable, then things do not work out as nicely. This is because the fundamental theorem of algebra is not true for multivariable polynomials. However, the following unique factorization theorem of polynomial algebra applies:

Unique Factorization. If f(x,y) is a polynomial of two variables with coefficients from $\mathbf{F}$, then $f(x,y) = \prod_{i=1}^{k} p_i(x,y)$ where the $p_i(x,y)$ are irreducible over $\mathbf{F}$.

Hence, if $\mathbf{X}$ is an m x n array and $\mathbf{t} \in C_{\mathbf{X}}$, then for any $\mathbf{z} \in \mathbf{X}$, $p_{\mathbf{t}}(x,y,\mathbf{z})$ can be thought of as an element of $\mathbf{F}[x,y]$ and can therefore be factored as $p_{\mathbf{t}}(x,y,\mathbf{z}) = \prod_{i=1}^{k} p_i(x,y)$. Since $p_{\mathbf{t}}$ has no power higher than m or n the factorization remains valid after making the substitutions $x^m=1$ and $y^n=1$. Thus $\mathbf{t} = \mathbf{t}_1 \oplus \mathbf{t}_2 \oplus \cdots \mathbf{t}_k \oplus \mathbf{s} = (\bigoplus_{i=1}^{k} \mathbf{t}_i) \oplus \mathbf{s}$ where the $\mathbf{t}_i$ are the templates corresponding to the p_i and $\mathbf{s}$ is a circular shift template.

Thus a nonseparable circulant template can be decomposed into a product of templates which are irreducible with respect to polynomial factorization. These irreducible factors can then possibly be handled by techniques such as matrix decompositions and/or

FFT's. Algorithms for factoring multivariable polynomials as well as a large number of references on the subject can be found in References 16 and 17

Since circulant templates compute circular convolutions, it is not surprising that polynomial algebra is helpful in designing parallel algorithms to compute them. Indeed, much of the theory of fast transforms, particularly the work of Winograd, References 18 and 19, and in Nussbaumer, References 20 and 19, is based on the use of the Chinese Remainder Theorem for polynomials to derive systematic methods for developing fast convolution algorithms. Our approach is fundamentally different. Whereas their criteria is the reduction of the number of arithmetic operations, particularly multiplications, our criteria is the decomposition of the templates required to execute a given transformation to a form compatible with a certain geometric configuration. We use the tool of polynomial factorization rather than the Chinese Remainder Theorem.

We have seen that these criteria can be compatible. Specifically, template decomposition usually does result in a reduced number of arithmetic operations. Thus the methodology of parallel algorithms via template decomposition and configuration restrictions often yields arithmetically efficient parallel algorithms.

5. PARALLEL ALGORITHMS IN THE IMAGE ALGEBRA

In this section specific applications of the general techniques described in the previous section are presented. The first is a class of algorithms for computing two-dimensional DFT's of arbitrary size on mesh connected arrays. The development of these algorithms is based on the use of the technique of template decomposition via matrix factorization. The second application is an algorithm for computing a global separable circulant template locally (with respect to mesh connected arrays) which uses Theorem 4.4.11. The template arises as the inverse of a separable blurring template. Thus this technique can be used to develop parallel algorithms for image restoration via inverse filtering when the degradation model is separable.

a. Two-Dimensional FFT's of Arbitrary Size for Mesh Connected Arrays

In this section we show how the relationship between the image algebra and linear algebra can be used to develop local, parallel algorithms for the two-dimensional FFT for images of arbitrary size. We will define the template used to compute the discrete Fourier transform (DFT). Since the DFT is separable the matrix corresponding to this template can

be factored into a Kronecker product of one dimensional Fourier matrices. These matrices can be factored into a product of tridiagonal matrices and permutation matrices. The permutations can be done in parallel by the odd-even transposition sort while the tridiagonal matrices are in fact local with respect to mesh connected arrays. Hence, translating the decompositions back into the image algebra will result in parallel algorithms. We illustrate this procedure for the case that **X** is a 100 x 100 array. Pease used matrix algebra to show how radix two FFT's could be implemented in parallel, Reference 21. Radix two FFT's for mesh connected arrays have also been developed, References 22 and 23.

b. Notation and Basic Properties

Unless otherwise stated we consider matrices and images as ordered from 0 to n-1. We denote by Σ_n the group of permutations of n objects, and by $\mathbf{Z}_n$ the ring of integers modulo n. We think of Σ_n as acting on $\mathbf{Z}_n$. Let $m \in \mathbf{Z}$ with $m > 1$ and let $\omega_m = \exp(-2\pi i/m)$ where $i = (-1)^{1/2}$.

Definition 4.5.1. Let $\sigma \in \Sigma_n$ and define the n x n matrix P_σ by

$$P_\sigma = (p_{ij}) \text{ where } p_{ij} = \begin{cases} 1 & \text{if } j = \sigma(i) \\ 0 & \text{otherwise} \end{cases}$$

P_σ is called a permutation matrix.

Assume for the rest of the section that $n = n_1 n_2$.

Definition 4.5.2. Define $\sigma_{n_1 n_2} : \mathbf{Z}_n \rightarrow \mathbf{Z}_n$ by the following rule:

If $i \in \mathbf{Z}_n$ and $i = a+bn_1$ with $0 \leq a < n_1$ and $0 \leq b < n_2$, then $\sigma_{n_1 n_2}(i) = an_2+b$.

$\sigma_{n_1 n_2}$ is called a generalized shuffle permutation.

Definition 4.5.3. Define $P(n_1,n_2) \in M_n$ by $P(n_1,n_2) = P_{\sigma_{n_1 n_2}}$. $P(n_1,n_2)$ is called a generalized shuffle permutation matrix.

Observe that $P(n_1,n_2) = P^t(n_2,n_1) = P^{-1}(n_2,n_1)$.

Definition 4.5.4. Let A be an n x m matrix and B an s x t matrix both with entries in **F**. Then the Kronecker, or Tensor Product of A and B, denoted $A \otimes B$, is the ns×mt matrix given by

$$A \otimes B = \begin{bmatrix} a_{0,0}B & \cdots & a_{0,n-1}B \\ \vdots & \ddots & \vdots \\ a_{m-1,0}B & \cdots & a_{m-1,n-1}B \end{bmatrix}$$

It is well known that $(A \otimes B)(C \otimes D) = AC \otimes BD$ provided that the matrices are all of the appropriate dimensions. Hence $\left(\prod_{i=0}^{k} A_i\right) \otimes I_j = \prod_{i=0}^{k} \left(A_i \otimes I_j\right)$. The Kronecker Product is not commutative. Rose has shown the following (Reference 24):

Fact: If A is an n x m matrix and B is a p x q matrix then $P(n,p)(A \otimes B)P(q,m) = B \otimes A$. In particular, if A is an n x n matrix and B is an m x m matrix, then $P(n,m)(A \otimes B)P(m,n) = B \otimes A$.

Definition 4.5.5. Define the n x n matrix $D(n_1,n_2)$ by

$$D(n_1,n_2) = \text{blockdiag}[D_k(n_1,n_2)],\ k=0,1,\ldots,n_2-1.$$

where

$$D_k(n_1,n_2) = \text{diag}(\omega_n^{ik}),\ i=0,1,\ldots,n_1-1,\ k=0,1,\ldots,n_2-1$$

For any $p,k > 1$ we denote $D_{p^k} \equiv D(p^{k-1},p)$ and $P_{p^k} \equiv P(p^{k-1},p)$.

Definition 4.5.6. Define the n x n matrix F_n by $F_n = n^{-1/2}(\omega_n^{ij})$. F_n is called the one-dimensional Fourier matrix of order n. If $\mathbf{x} \in \mathbf{F}^n$ then $F_n\mathbf{x}$ is the one-dimensional discrete Fourier transform (1-dim DFT) of **x**.

Definition 4.5.7. Let **X** be an m x n array . Define $\mathbf{g} \in \mathbf{F}^{\mathbf{X} \to \mathbf{X}}$ by $\mathcal{G}(u,v) = \mathbf{X}$ and $g_{(u,v)} = (nm)^{-1/2}\exp[-2\pi i(uj/n+vk/m)]$, where $\mathbf{j},\mathbf{k} \in \mathbf{F}^{\mathbf{X}}$ are defined by $\mathbf{j} = \{ (x,y),j(x,y)) : j(x,y) = x \}$ and $\mathbf{k} = \{ (x,y),k(x,y)) : k(x,y) = y \}$. **g** is the two dimensional DFT template.

The transform $\mathbf{a} \mapsto \mathbf{a} \oplus \mathbf{g}$ is the DFT of $\mathbf{a}$.

Definition 4.5.8. Let $\mathbf{X}$ be an m x n array and let $\mathbf{g}$ be as in the previous definition. Then let $F_{mxn} = \Psi(\mathbf{g})$. F_{mxn} is called the *two-dimensional Fourier matrix* of order m x n.

Notice that $F_{nxm} \neq F_{nm}$ in general. Notice also that $F_{mxn} = F_m \otimes F_n = (F_m \otimes I_n)(I_m \otimes F_n)$. This last observation is an equation which expresses the fact that the two dimensional DFT can be computed by first computing one dimensional DFT's along each row and then along each column of the result. A tridiagonal matrix computes a linear transformation of a linear array locally. Permutations of linear arrays can be computed locally using the odd-even transposition sort, Reference 25. Hence if tridiagonal decompositions of the one dimensional Fourier matrix can be found for every n then local DFT's could be implemented on mesh connected arrays. These decompositions have been derived. We present them in the next section.

c. Tridiagonal Decompositions of the Fourier Matrices

In this section we present the sequence of theorems resulting in the method for factoring the Fourier matrices into products of tridiagonal matrices and permutation matrices. The proofs can be found in Reference 6. It is our intention here to show how they can be used along with the image algebra to derive algorithms for the local implementation of FFT's on mesh connected arrays. The statements of the theorems show how to proceed and therefore we present them.

Assume that $n = mk$. Rose, Reference 24, has shown that

$$F_n = (F_m \otimes I_k) D(k,m) (I_m \otimes F_k) P(k,m)$$

and, that if m and k are relatively prime, then there exists permutation matrices Q_1 and Q_2 such that

$$F_n = Q_1 (F_m \otimes F_k) Q_2$$

Using these results one can prove the following theorems:

Theorem 4.5.9. Suppose $n = \prod_{i=1}^{s} p_i^{k_i}$ with $s \geq 2$. Let

$$n_j = \begin{cases} \prod_{i=1}^{j} p_i^{k_i} & \text{if } 1 \le j \le s \\ 1 & \text{if } j = -1,0 \end{cases} \text{, and } c_j = n/n_j \text{ for } -1 \le j \le s,$$

and $q_j = p_j^{k_j}$ for $1 \le j \le s$, $q_0 = 1$. Then there exists permutation matrices Q_0, Q_1,..., Q_{s-2}, H such that

$$F_n = \left[\prod_{j=0}^{s-2} Q_j (I_{n_j} \otimes F_{q_{j+1}} \otimes I_{c_{j+1}})\right](I_{n_{s-1}} \otimes F_{q_s})H$$

Theorem 4.5.10. $F_{p^k} = G_p(I_{p^{k-1}} \otimes F_p)H_p$, where

$$G_p = \prod_{m=0}^{k-2} \left[(I_{p^m} \otimes P_{p^{k-m}})(I_{p^{k-1}} \otimes F_p)(I_{p^m} \otimes P^t_{p^{k-m}})(I_{p^m} \otimes D_{p^{k-m}})\right]$$

and

$$H_p = \prod_{m=2}^{k} \left[I_{p^{k-m}} \otimes P_{p^m}\right].$$

The previous theorems are matrix identities associated with FFT's. When the blocklength is prime the FFT no longer works. This is one reason why the radix two case is much easier to develop a parallel algorithm for. Indeed, the last theorem, with $p = 2$, is a tridiagonal decomposition of power of two Fourier matrices and therefore yields a local algorithm. For the general case we resort to other matrix identities. These identities are based on the Rader prime algorithm, Reference 26, and the circular convolution theorem in matrix form.

Some notation is required. Let A be an n x n matrix. We denote by $A^{(m)}$ the (n+m) x (n+m) matrix

$$A^{(m)} = \begin{bmatrix} I_m & 0 \\ 0 & A \end{bmatrix}$$

We denote by U_p the p x p matrix with all one's in the first column and one's down the diagonal. For example if $p = 5$ then

$$U_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Let p be a prime. Then it is well known that for every positive integer α with $1 < \alpha < p$, p-1 is the smallest positive integer such that $\alpha^{p-1} \pmod p = 1$. Choose any such α and let $c_i = \omega_p^{\alpha^{p-i-1}} - 1$. Define the polynomial $f_p(x) = c_0 + c_1 x + \cdots + c_{p-2}x^{p-2}$. Then define the p x p matrix Λ_p by $\Lambda_p = \text{diag}(1, f_p(\omega_{p-1}^{i-1}))$, i=1,2,...,p-1.

Theorem 4.5.11. (Prime Decomposition Theorem) Suppose p is a prime. Then there exists permutation matrices R_1 and R_2 such that

$$F_p = U_p R_1 F_{p-1}^{(1)} \Lambda_p F_{p-1}^{*(1)} R_2 U_p^t$$

where * denotes the conjugate transpose.

Observe that if A and B are any n x n matrices then $(I_m \otimes A^*) = (I_m \otimes A)^*$ and $A^{(m)}B^{(m)} = (AB)^{(m)}$. Therefore if $F_{p-1}^{(1)} = \prod T_i$ where the T_i's are tridiagonal matrices then $I_m \otimes F_{p-1}^{(1)} = I_m \otimes (\prod T_i)^{(1)} = I_m \otimes \prod (T_i^{(1)}) = \prod (I_m \otimes T_i^{(1)})$ is a product of tridiagonals. Thus the tridiagonalization problem is reduced to factoring matrices of the form U_p and $F_{p-1}^{(1)}$. But $F_{p-1}^{(1)} = \text{blockdiag}[1, F_{p-1}]$ so it is clear that one can proceed inductively until p=2 or p=3 in order to factor $F_{p-1}^{(1)}$. It is easy to see how to factor U_p by considering an example.

Suppose that $p = 5$. Then if $\mathbf{x} = (x_0, x_1, x_2, x_3, x_4)^t$, $U_5\mathbf{x} = (x_0, x_0+x_1, x_0+x_2, x_0+x_3, x_0+x_4)$. The algorithm depicted in Figure 18 will compute $U_p\mathbf{x}$ locally in p-1 steps. In matrix form

$$U_p = \left(\prod_{k=p-1}^{2} \begin{bmatrix} I_k & 0 & 0 \\ 0,\ldots,0,1 & 1 & 0 \\ 0 & 0 & I_{p-k-1} \end{bmatrix} \right) \begin{bmatrix} 1 & 0 & . & . & . & 0 & 0 \\ 1 & 1 & 0 & . & . & . & . \\ 0 & -1 & 1 & . & . & . & . \\ . & 0 & -1 & . & . & . & . \\ . & . & 0 & . & . & 0 & . \\ . & . & . & . & . & 1 & 0 \\ 0 & 0 & 0 & . & . & -1 & 1 \end{bmatrix}$$

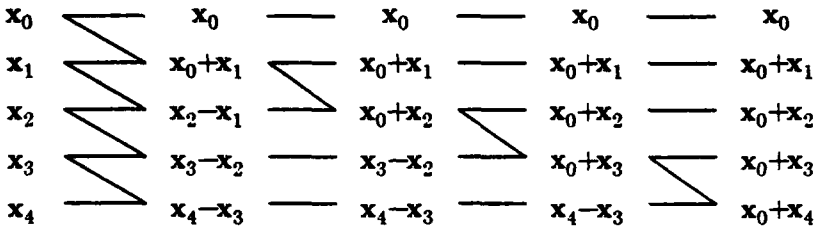


Figure 18. Local Algorithm for Computing $U_5\mathbf{x}$

Hence the problem is successively reduced from F_n to F_{p^k} to F_p to F_{p-1}. This reduction process terminates when p=2 or p=3 in which case we have all tridiagonal or permutation factors. We now show how these theorems can be used.

d. Permutations

In this section we describe how separable permutation mappings of rectangular images can be expressed as sequences of local permutations, or transpositions, in the image algebra using the odd-even transposition sort (OETS). These types of permutations are required for implementing the DFT locally.

Definition 4.5.12. A permutation template $\mathbf{t}$ is an template with the properties that for every $\mathbf{x} \in \mathbf{X}$, $|T(\mathbf{x})| = 1$ and if $\mathbf{y} \in T(\mathbf{x})$, then $t_x(\mathbf{y}) = 1$.

If $\mathbf{t}$ is a permutation template then $\mathbf{a} \oplus \mathbf{t}$ is an image with the same grey values as $\mathbf{a}$ but (possibly) in different locations. Henceforth we assume that $\mathbf{X}$ is an m x n array.

Definition 4.5.13. A row separated permutation template is a permutation template $\mathbf{t}$

with the property that $t_{(x,y)}(u,v) = 1$ implies that $u = x$.

A row separated template will permute each row separately and simultaneously but possibly with different permutations. These types of permutations are actually more general than are required for the local DFT but the OETS algorithm will implement them just as easily as the more restrictive case. If $\mathbf{t}$ is as in Definition 4.5.13 then the mapping $\mathbf{t} \mapsto \mathbf{a} \oplus \mathbf{t}$ is called a row separated permutation. The same definitions and remarks can be made for the columns.

The OETS applied to the y^{th} row of an image is given by the following rules:

Alternate the following steps:

(1) if y is odd and $\mathbf{a}(x,y) < \mathbf{a}(x,y+1)$, then switch $\mathbf{a}(x,y)$ and $\mathbf{a}(x,y+1)$, and

(2) if y is even and $\mathbf{a}(x,y) < \mathbf{a}(x,y+1)$, then switch $\mathbf{a}(x,y)$ and $\mathbf{a}(x,y+1)$.

The output will be an image with each row sorted in increasing order. It can be shown that the OETS will take at most n parallel steps to execute where i. and ii. each count as one step, Reference 25.

We now write the OETS in the image algebra. The idea is that, for a given permutation an image $\mathbf{d}$ will be defined containing the inverse of the permutation in each row. This is assumed to have been done. The sequence of transpositions required to sort $\mathbf{d}$ will be applied to an input image $\mathbf{a}$. The result will be that the rows of $\mathbf{a}$ are permuted by the given permutation.

For example, if

$$\mathbf{a} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \quad \text{and} \quad \mathbf{d} = \begin{bmatrix} 3 & 1 & 2 & 0 \\ 0 & 2 & 3 & 1 \\ 0 & 2 & 1 & 3 \end{bmatrix}$$

then the output image will be

$$\begin{bmatrix} 4 & 2 & 3 & 1 \\ 5 & 7 & 8 & 6 \\ 9 & 11 & 10 & 12 \end{bmatrix}$$

The first step is to define the required templates. In what follows, we do not explicitly define the configurations. If a template in question has nonzero weights at a point, that is, if $(x,y) \in \mathbf{X}$ such that there is a $(u,v) \in \mathbf{X}$ with $t_{(x,y)}(u,v) \neq 0$, then we take the configuration at (x,y) to be all such points (u,v). If not, then we take the configuration at the point (x,y) to be the set $\{(x,y)\}$. The template definitions are then given by:

$\mathbf{t_o}$:

If y is odd and $y \neq n-1$, then for every x

$$t^{o}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y+1) \\ 0 & \text{else} \end{cases}$$

If y is even and $y \neq 0$, then for every x

$$t^{o}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y-1) \\ 0 & \text{else} \end{cases}$$

Otherwise for every x and (u,v), $t_{(x,y)}(u,v) = 0$.

$\mathbf{t_e}$:

If y is odd, then for every x

$$t^{e}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y-1) \\ 0 & \text{else} \end{cases}$$

If y is even and $y \neq n-1$, then

$$t^{e}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y+1) \\ 0 & \text{else} \end{cases}$$

If y is even and $y = n-1$, then for every x and (u,v), $t_{(x,y)}(u,v) = 0$.

$\mathbf{s_o}$:

If y is odd and $y \neq 0, n-1$, then for every x

$$s^{o}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ -1 & \text{if } (u,v) = (x,y+1) \\ 0 & \text{else} \end{cases}$$

Otherwise $s^{o}_{(x,y)}(u,v) = 0$.

$\mathbf{s}_e$:

If y is even and $y \neq n-1$, then for every x

$$s^{e}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ -1 & \text{if } (u,v) = (x,y+1) \\ 0 & \text{else} \end{cases}$$

Otherwise for every x and (u,v), $s^{e}_{(x,y)}(u,v) = 0$.

$\mathbf{r}_o$:

If $y \neq 0$, then for every x

$$r^{o}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y-1) \\ 0 & \text{else} \end{cases}$$

Otherwise for every x and (u,v), $r^{o}_{(x,y)}(u,v) = 0$.

$\mathbf{r}_e$:

If $y \neq 0$, then for every x

$$r^{e}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y-1) \\ 0 & \text{else} \end{cases}$$

If $y = 0$, then for every x

$$r^{e}_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 0 & \text{else} \end{cases}$$

Otherwise for every x and (u,v), $r^{e}_{(x,y)}(u,v) = 0$.

The templates $\mathbf{t}_o$, $\mathbf{s}_o$, and $\mathbf{r}_o$ are depicted in Figure 19. The others are similar.

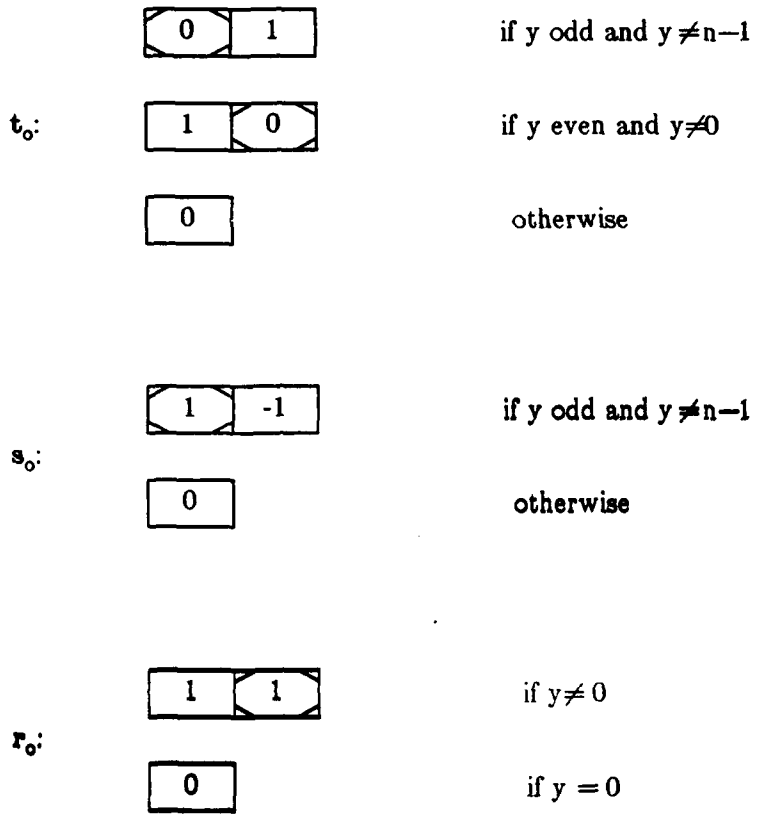


Figure 19. Templates Used to Implement OETS

The OETS algorithm takes the form:

Input image **a**

DO 5 i = 1,nmax

DO 5 j = 1,2

IF j = 1 **THEN** v = o **ELSE** v = e

$\mathbf{b} = \chi_{>0}(\mathbf{d} \oplus \mathbf{s}_v)$

$\mathbf{e} = \mathbf{b} \oslash \mathbf{r}_v$

$\mathbf{c} = \chi_{<1}(\mathbf{e})$

$\mathbf{b} = \mathbf{d}^* \mathbf{c} + \mathbf{e}^* (\mathbf{d} \oplus \mathbf{t}_v)$

$\mathbf{d} = \mathbf{a}^*\mathbf{c} + \mathbf{e}^*(\mathbf{a} \oplus t_v)$

$\mathbf{a} = \mathbf{d}$

$\mathbf{d} = \mathbf{b}$

5 **CONTINUE**

Output the permuted image **a**

Note that due to the use of the symbol = to denote equality rather that assignment we do not write statements of the form $\mathbf{a} = \mathbf{a}^*\mathbf{c}$. The parameter nmax can be set to the maximum number of steps that the OETS will take. For an arbitrary permutation nmax = n. However, if n = mk, where m,k > 1, then it can be shown that the OETS will execute the generalized shuffle permutations in at most (m-1)(k-1)+1 steps, Reference 27. Since each run through the inner loop represents one step one can set nmax = $\lceil (m-1)(k-1)/2 \rceil$.

e. Local Implementation of the 100 x 100 DFT

In this section an algorithm for the implementation of a 100 x 100 DFT on a mesh connected array using the image algebra is developed. We are assuming that the array is at least 100 x 100. As pointed out earlier, the problem is to derive a tridiagonal/permutation decomposition of the matrix F_{100}. We do so and then interpret the result as a template/image decomposition. We write the algorithm in image algebra pseudo-code and we count the overall number of parallel steps required. We assume that the permutations are precomputed and executed by the routine given in the previous section.

Using the decomposition theorems the following equations can be derived: There exist permutation matrices Q_1, Q_2 such that

$$F_{100} = Q_1(F_4 \otimes I_{25})(I_4 \otimes F_{25})Q_2.$$

Since

$$F_4 = (F_2 \otimes I_2)D(2,2)(I_2 \otimes F_2)P(2,2)$$

it is clear that

$$F_4 \otimes I_{25} = P(50,2)(I_{50} \otimes F_2)P(2,50)(D(2,2) \otimes I_{25})(I_2 \otimes P(25,2))(I_{50} \otimes F_2)(I_2 \otimes P(2,25))(P(2,2) \otimes I_{25}).$$

All the matrices on the right hand side of the last equation are either tridiagonal or permutation matrices. As for the other factor

$$F_{25} = P(5,5)(I_5 \otimes F_5)P(5,5)D(5,5)(I_5 \otimes F_5)P(5,5).$$

Using the prime decomposition theorem, there exists permutation matrices R_1, R_2 such that

$$F_5 = U_5 R_1 F_4^{(1)} \Lambda_5 F_4^{*(1)} R_2 U_5^t$$

where the α needed to define Λ can be taken to be 2, 3, or 4. Continuing

$$F_4^{(1)} = (F_2 \otimes I_2)^{(1)} D(2,2)^{(1)} (I_2 \otimes F_2)^{(1)} P(2,2)^{(1)}.$$

Furthermore

$$(F_2 \otimes I_2)^{(1)} = P(2,2)^{(1)} (I_2 \otimes F_2)^{(1)} P(2,2)^{(1)}$$

Hence, since $(F_4^*)^{(1)} = (F_4^{(1)})^*$, we can put these equations together to write down a tridiagonal/permutation decomposition of $I_4 \otimes F_{25}$. Thus we arrive at a decomposition of F_{100}. In order to compute the DFT locally along each row, the transformation corresponding to the matrix $I_{100} \otimes F_{100}$ must be implemented locally. This can be done by implementing the sequence of transformations indicated in Figure 20. Since all matrices appearing in the table are tensored on the left by I_{100} the notation is suppressed. It can be seen that the total number of parallel steps is, at most, 1014 permutation steps, 52 addition steps with at most one addition per pixel, and 18 multiplication steps with at most one complex multiplication per pixel. The column transforms are treated similarly.

Matrix	Type	# of Parallel Steps	Operations/Pixel/Step
1.) $(I_4 \otimes P(5,5))Q_2$	P	100	
2.) $I_{20} \otimes U_5^t$	A	4	1
3.) $I_{20} \otimes P(2,2)^t$	P	1	
4.) $I_{20} \otimes R_2$	P	4	
5.) $I_{20} \otimes (I_2 \otimes F_2)^{(1)}$	A	1	1
6.) $I_{20} \otimes D(2,2)^{*(1)}$ or	M	1	1
6.) $I_{20} \otimes D(2,2)^{(1)}$	M	1	1
7.) $I_{20} \otimes P(2,2)^{(1)}$	P	1	
8.) $I_{20} \otimes (I_2 \otimes F_2)^{(1)}$	A	1	1
9.) $I_{20} \otimes P(2,2)^{(1)}$	P	1	
10.) $I_{20} \otimes \Lambda_5$	M	1	1
11.) Repeat 5-9			
12.) $I_{20} \otimes R_1$	P	5	
13.) $I_{20} \otimes U_5$	A	4	1
14.) $I_4 \otimes D(5,5)$	M	1	1
15.) $I_4 \otimes P(5,5)$	P	17	
16.) Repeat 2-15			
17.) $I_4 \otimes P(5,5)$	P	17	
18.) $P(2,2) \otimes I_{25}$	P	100	
19.) $I_2 \otimes P(2,25)$	P	24	
20.) $I_{50} \otimes F_2$	A	1	1
21.) $I_2 \otimes P(25,2)$	P	24	
22.) $D(2,2) \otimes I_{25}$	M	1	1
23.) $P(2,50)$	P	100	
24.) $I_{50} \otimes F_2$	A	1	1
25.) $Q_1 P(50,2)$	P	100	

Figure 20. Sequences of Local Transformations Required to Compute the Row transforms of a 100× 100 DFT. P stands for addition and M for multiplication.

We now illustrate how to use the operation preserving mapping Ψ^{-1} to write the algorithm in the image algebra. We first define the required templates and then the required images (recall that a diagonal matrix corresponds to an image). In these template definitions, we take the configuration at a point (x,y) to be the set of all points (u,v) such that $t_{(x,y)}(u,v) \neq 0$.

The template t corresponding to $I_{100} \otimes (I_2 \otimes F_2)^{(1)}$ is given by:

If y mod 3 = 0, then for every x

$$t_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 0 & \text{else} \end{cases}$$

If y mod 3 = 1, then for every x

$$t_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y+1). \\ 0 & \text{else} \end{cases}$$

If y mod 3 = 2, then for every x

$$t_{(x,y)}(u,v) = \begin{cases} -1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y-1). \\ 0 & \text{else} \end{cases}$$

The template s corresponding to $I_{100} \otimes I_{50} \otimes F_2$ is given by:

If y is even, then for every x

$$s_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y+1). \\ 0 & \text{else} \end{cases}$$

If y is odd, then for every x

$$s_{(x,y)}(u,v) = \begin{cases} -1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y-1). \\ 0 & \text{else} \end{cases}$$

The templates corresponding to the decomposition of $I_{100} \otimes I_{20} \otimes U_5$ are given by:

t_1:

If y mod 5 = 0, then for every x

$$t^1_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 0 & \text{else} \end{cases}$$

If y mod 5 = 1, then for every x

$$t^1_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y-1). \\ 0 & \text{else} \end{cases}$$

Otherwise for every x

$$t^1_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ -1 & \text{if } (u,v) = (x,y-1). \\ 0 & \text{else} \end{cases}$$

t_i, i = 2,3,4:

If y mod 5 = i, then for every x

$$t^i_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y-1). \\ 0 & \text{else} \end{cases}$$

otherwise for every x

$$t^i_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 0 & \text{else} \end{cases}$$

$t_1 \oplus t_2 \oplus t_3 \oplus t_4$ will correspond to $I_{100} \otimes I_{20} \otimes U_5$. The templates t^t_1, t^t_2, t^t_3, t^t_4 corresponding to $I_{100} \otimes I_{20} \otimes U^t_5$ are defined in an analogous fashion. The preceding templates will compute all the additions required. Note that the configurations associated with the above templates have at most two elements each.

The images corresponding to the diagonal matrices are now defined. These images will be involved in computing all the multiplications.

The image **c** corresponding to $I_{100} \otimes I_{20} \otimes \Lambda_5$ is defined first. Recalling the definition of Λ_5 it can be seen that

$$\mathbf{c} = \{ (x,y,\mathbf{c}(x,y)) : \mathbf{c}(x,y) = \begin{cases} \lambda_k & \text{if } y \bmod 5 = k \neq 0 \\ 1 & \text{else} \end{cases} \}$$

where $\lambda_k = f_4(i^{k-1})$, $i = (-1)^{1/2}$. Defining **c** in terms of the image algebra requires no if-then statements:

$$\mathbf{c} = \sum_{j=0}^{4} \chi_{-j}(\mathbf{k} \bmod 5) * \lambda_j$$

where **k** is the image used to define the DFT template and $\lambda_0 \equiv 1$.

The images $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ corresponding to $I_{100} \otimes D(2,2) \otimes I_{25}$, $I_{100} \otimes I_4 \otimes D(5,5)$, and $I_{20} \otimes D(2,2)^{(1)}$ respectively are now defined. Note that $\omega_4 = -i = -(-1)^{1/2}$. Hence $D(2,2) = \text{diag}(1,1,1,-i)$ and $D(2,2) \otimes I_{25} = \text{blockdiag}[I_{25}, I_{25}, I_{25}, -iI_{25}]$. Thus

$$\mathbf{b}_1 = \chi_{\geq 75}(\mathbf{k}) * (-i) + \chi_{< 75}(\mathbf{k})$$

Let $\omega \equiv \omega_{25}$. Observe that

$$D(5,5) = \text{diag}(1,1,1,1,1,1,\omega,\omega^2,\omega^3,\omega^4,1,\omega^2,\omega^4,\omega^6,\omega^8,1,\omega^3,\omega^6,\omega^9,\omega^{12},1,\omega^4,\omega^8,\omega^{12},\omega^{16}).$$

Hence, defining the images **r** and **q** by $\mathbf{r} = \mathbf{k} \bmod 5$ and $\mathbf{q} = \text{INT}((\mathbf{k}-\mathbf{r})/5)$, where INT(x) denotes the integer part of x, we have

$$\mathbf{b}_2 = (\omega_{25} * 1) ** (\mathbf{r} * \mathbf{q}).$$

Similarly,

$$\mathbf{b}_3 = \chi_{-4}(\mathbf{r}) * (-i) + \chi_{\neq 4}(\mathbf{r})$$

The image corresponding to the conjugate is defined by replacing i with -i.

This completes the definitions of the images and templates used to compute the row transforms. Those required to compute the column transforms can easily be defined by interchanging the roles of x and y and **j** and **k**. The code for the column transforms can also be

integrated into the code for the row transforms by adding a few if-then-else statements. We now write the algorithm in the image algebra. The notation **PERMUTE(a,b)** denotes a call to the permutation subroutine where the image **a** is permuted and the result is put into **b**. Notice that due to the use of the symbol = to denote equality rather than assignment we do not write statements of the form **a** = **a***b.

Input 100 x 100 image **a**

PERMUTE(a,b)

$\mathbf{a} = \mathbf{b} \oplus t_4^t \oplus t_3^t \oplus t_2^t \oplus t_1^t$

PERMUTE(a,b)

r = **k** mod 5

q = INT((**k**-**r**)/5)

$\mathbf{b}_1 = \chi_{\geq 75}(\mathbf{k})*(-i) + \chi_{<75}(\mathbf{k})$

$\mathbf{b}_2 = (\omega_{25}*1)**(\mathbf{r}*\mathbf{q})$

$\mathbf{b}_3 = \chi_{=4}(\mathbf{r})*(-i) + \chi_{\neq 4}(\mathbf{r})$

$\mathbf{b}_3^* = \chi_{=4}(\mathbf{r})*i + \chi_{\neq 4}(\mathbf{r})$

DO 10 m = 1,2

DO 5 n = 1,2

PERMUTE(b,a)

$\mathbf{b} = \mathbf{a} \oplus t$

IF n = 1 **THEN**

$\mathbf{a} = \mathbf{b}*\mathbf{b}_3^*$

ELSE

$\mathbf{a} = \mathbf{b}*\mathbf{b}_3$

ENDIF

PERMUTE(a,b)

IF n = 1 **THEN**

a = b*c

b = a

ENDIF

5 **CONTINUE**

PERMUTE(b,a)

b = a⊕t_1⊕t_2⊕t_3⊕t_4

a = b*b_2

PERMUTE(a,b)

10 **CONTINUE**

PERMUTE(b,a)

b = a⊕s

PERMUTE(a,b)

b = a*b_1

PERMUTE(b,a)

b = a⊕s

PERMUTE(b,a)

Output **a**, the row transforms are complete.

Algorithms for implementing the two dimensional DFT of any size can be derived using this methodology. If the array size is not as large as the image size but the image dimensions factor into products of integers smaller than the array size then judicious manipulation of the matrix identities can lead to algorithms for breaking the computation into a succession of smaller two-dimensional DFT's. For example suppose a 1000 x 1000 DFT is required on a 100 x 100 mesh connected array. Then since

$$F_{1000\times1000} = F_{1000}\otimes F_{1000} =$$

$$= \left[F_{(10}\otimes I_{100})D(10,100)(I_{10}\otimes F_{100})P(10,100)\right]\otimes\left[(F_{10}\otimes I_{100})D(10,100)(I_{10}\otimes F_{100})P(10,100)\right] =$$

$$= \left[(F_{10}\otimes I_{100})\otimes(F_{10}\otimes I_{100})\right]\left(D(100,10)\otimes D(100,10)\right)\left[(I_{10}\otimes F_{100})\otimes(I_{10}\otimes F_{100})\right]$$

$$* \left(P(100,10)\otimes P(100,10)\right) = P_1(I_{10000}\otimes F_{10\times10})P_2DP_3(I_{100}\otimes F_{100\times100})P_4.$$

where the P_i are permutation matrices and D is a diagonal matrix. Thus the 1000 x 1000 can be broken down into 100 100 x 100 DFT's and 10,000 10 x 10 DFT's. One could also use 25 and 16 as the factors rather than 10 and 100. In any case, DFT's of size larger than the array size can be handled in this way.

By the convolution theorem this technique can be used to compute any circulant transform locally. Thus the methods presented here offer two alternative techniques for computing separable circulant transforms locally.

f. Local Inverse Templates

We now present an example of how Theorem 4.4.11 can be used to develop parallel algorithms. Let $\mathbf{X}$ be an 8 x 8 array and let $\mathbf{m} \in F_{X\to X}$ be the circulant local average template, i.e. $m_y(x) = \frac{1}{9}$ if x is an eight neighbor of y and 0 else and $\mathbf{m}$ wraps around at the boundary. It can be verified by direct calculation that $\mathbf{m}$ is invertible with circulant inverse $\mathbf{m}^{-1} = \mathbf{n}$ shown below. Thus $(\mathbf{a}\oplus\mathbf{m})\oplus\mathbf{m}^{-1} = \mathbf{a}$. In particular, we can apply the local averaging transform $(\mathbf{a}\oplus\mathbf{m})$ and then recover the original image $\mathbf{a}$ by applying the template $\mathbf{m}^{-1}$ to the averaged image, or we can first do the transform $\mathbf{a}\oplus\mathbf{m}^{-1}$ to obtain the camouflaged image shown in the bottom left-hand corner in Figure 21, and then apply the local averaging filter to this transform in order to recover $\mathbf{a}$.

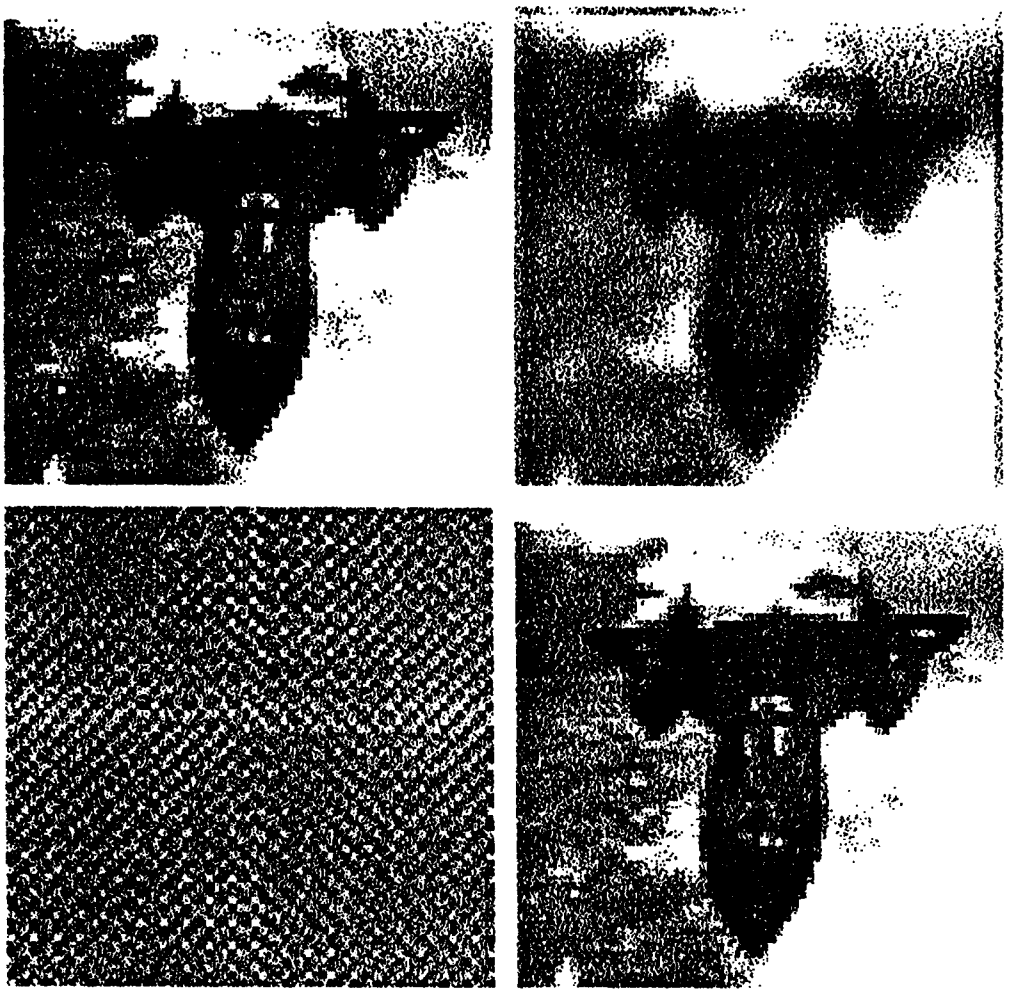


Figure 21. Inverse Averaging Example

The upper left image in Figure 21 is the source image **a**; the upper right image is the averaged image, $\mathbf{a} \oplus \mathbf{m}$; the lower left image $\mathbf{a} \oplus \mathbf{m}^{-1}$; and the lower right image is $(\mathbf{a} \oplus \mathbf{m}) \oplus \mathbf{m}^{-1} = (\mathbf{a} \oplus \mathbf{m}^{-1}) \oplus \mathbf{m} = \mathbf{a}$, that is, the average of the inverse average of image **a**.

As shown in Figure 22, the template **n** is a global template in the sense that the new value at a point depends on the value at every other point.

$\mathbf{n} =$

/1/	-2	1	1	-2	1	1	-2
-2	4	-2	-2	4	-2	-2	4
1	-2	1	1	-2	1	1	-2
1	-2	1	1	-2	1	1	-2
-2	4	-2	-2	4	-2	-2	4
1	-2	1	1	-2	1	1	-2
1	-2	1	1	-2	1	1	-2
-2	4	-2	-2	4	-2	-2	4

Figure 22. Inverse of the Local Averaging Filter on an 8×8 Array

However, $\mathbf{n}$ is separable with $\mathbf{n} = \mathbf{s} \oplus \mathbf{t}$. The templates $\mathbf{s}$ and $\mathbf{t}$ are shown in Figure 23 below.

$\mathbf{s} =$

/1/	-2	1	1	-2	1	1	-2

$\mathbf{t} =$

/1/
-2
1
1
-2
1
1
-2

Figure 23. Separable Components of the Inverse Averaging Filter $\mathbf{n}$

We show how $\mathbf{t}$ can be decomposed into local templates, the decomposition of $\mathbf{s}$ being similar. Since $\mathbf{t}$ is separable and circulant, Theorem 4.4.11 applies. The point 0 is a

minimal point for **t**. The polynomial representative of **t** at 0 is given by

$$p_t(x,y,0) = 1-2x+x^2+x^3-2x^4+x^5+x^6-2x^7.$$

The roots of $p_t(x,y,0)$ are:

$$q_1 = 0.749927035578$$

$$q_2, q_3 = 0.678637719963 \pm 0.47903181019i$$

$$q_4, q_5 = -1.00449681937 \pm 0.3532353691i$$

$$q_6, q_7 = 0.200895581622 \pm 0.901038638456i$$

where $i = (-1)^{1/2}$. Hence, if **a** is an arbitrary image on **X** then $\mathbf{a} \oplus \mathbf{t}$ can be computed locally as $-2 * ((((((\mathbf{a} \oplus \mathbf{q}_1) \oplus \mathbf{q}_2) \oplus \mathbf{q}_3) \cdots) \oplus \mathbf{q}_7)$, where the $\mathbf{q}_i$ are of the form indicated in Figure 17.

6. LATTICE STRUCTURES IN THE IMAGE ALGEBRA

The next sections discuss the lattice substructures inherent in the Image Algebra. Investigation of properties involving $\bigvee$, $\vee$, and $\boxed{\vee}$ has lead to the identification of lattice structures and some useful applications. Discussed here are the vector lattice of images and the structures of templates under $\bigvee$ and $\boxed{\vee}$. The relation of mathematical morphology to the Image Algebra is discussed. Properties involving images and templates are also presented.

a. The Vector Lattice of Images

Let **X** be a coordinate set with a finite number of elements, $\mathbf{X} = \{ \mathbf{x}_1, \ldots, \mathbf{x}_n \}$. Define $\psi : (\mathbf{R})^{\mathbf{X}} \rightarrow \mathbf{R}^n$ by $\mathbf{a} \mapsto v_{\mathbf{a}}$, where $v_{\mathbf{a}} = (a(x_1),\ldots,a(x_n))$.

The function ψ is an isomorphism from $\mathbf{R}^{\mathbf{X}}$ to the real vector space $\mathbf{R}^n$. It is a trivial matter to show that ψ is one-to-one and onto; to show that $\psi(\mathbf{a} + \mathbf{b}) = \psi(\mathbf{a}) + \psi(\mathbf{b})$, we see that for $\mathbf{c} = \mathbf{a} + \mathbf{b}$, $\psi(\mathbf{c}) = (\mathbf{a}(\mathbf{x}_1) + \mathbf{b}(\mathbf{x}_1),\ldots,\mathbf{a}(\mathbf{x}_n) + \mathbf{b}(\mathbf{x}_n)) = v_{\mathbf{a}} + v_{\mathbf{b}} = \psi(\mathbf{a}) + \psi(\mathbf{b})$, using the definition of vector addition. The remaining properties, namely $\psi(\alpha\mathbf{a}) = \alpha\psi(\mathbf{a})$, $\alpha \in \mathbf{R}$, and $\psi(\mathbf{a} \bullet \mathbf{b}) = \psi(\mathbf{a}) \bullet \psi(\mathbf{b})$ are similarly easy to prove.

Definition. A partly ordered vector space V is a partially ordered, additive (abelian) group in which $x \geq 0$ in V and $\lambda \geq 0$ in **R** imply $\lambda x \geq 0$ in V.

Definition. A vector lattice is a partly ordered vector space which is also an l-group.

For example, for any cardinal number n, $\mathbf{R}^n$ is a vector lattice. Thus, we have that $\mathbf{R}^{\mathbf{X}}$ is a vector lattice, and any results known about vector lattice are directly applicable to

the corresponding structure of images. A few properties are listed here.

$$\mathbf{a}+(\mathbf{b}\vee\mathbf{c})=(\mathbf{a}+\mathbf{b})\vee(\mathbf{a}+\mathbf{c}),\ \forall\,\mathbf{a},\,\mathbf{b},\,\mathbf{c}\in\mathbf{R}^{\mathbf{X}}. \tag{10}$$

$$|\mathbf{a}+\mathbf{b}|\leq|\mathbf{a}|+|\mathbf{b}|,\ \text{where}\ |\mathbf{a}|=\mathbf{a}\vee\mathbf{0}-\mathbf{a}\wedge\mathbf{0}>\mathbf{0}\ \text{unless}\ \mathbf{a}=\mathbf{0},\ \forall\,\mathbf{a},\,\mathbf{b}\in\mathbf{R}^{\mathbf{X}}. \tag{11}$$

7. THE ALGEBRA DETERMINED BY $\textcircled{\vee}$

a. Operations Between Templates under $\textcircled{\vee}$

Most results presented here are for the general case of templates from $\mathbf{Y}$ to $\mathbf{X}$, though of course they hold for the specific case when $\mathbf{Y}=\mathbf{X}$. We will also discuss the concept of equivalence classes of templates which is used to describe the structures identified.

Let $\mathrm{i}\in\mathbf{R}$, and define $\mathbf{R}^{>\mathrm{i}}_{\mathbf{Y}\to\mathbf{X}}=\{\mathbf{t}\in\mathbf{R}_{\mathbf{Y}\to\mathbf{X}}\colon t_{\mathbf{y}}(\mathbf{x})>\mathrm{i}\ \text{if}\ \mathbf{x}\in T(\mathbf{y})\}$ and $\mathbf{R}^{\mathbf{X}}_{\geq 0}=\{\mathbf{a}\in\mathbf{R}^{\mathbf{X}}\colon\mathbf{a}\geq 0\}$.

We begin with a basic identity.

Lemma 4.7.1. $\mathbf{s}\,\textcircled{\wedge}\,\mathbf{t}=-(\mathbf{s}\,\textcircled{\vee}\,(-\mathbf{t}))$, for $\mathbf{s}\in\mathbf{R}^{>0}_{\mathbf{Z}\to\mathbf{X}}$, $\mathbf{t}\in\mathbf{R}^{>0}_{\mathbf{Y}\to\mathbf{Z}}$.

Proof: Both templates are members of $\mathbf{R}^{>0}_{\mathbf{Y}\to\mathbf{X}}$, and have the same configuration. As for the gray level values, on the left hand side of the equality the image at location $\mathbf{y}\in\mathbf{Y}$ has form $\{(\mathbf{x},r_{\mathbf{y}}(\mathbf{x}))\colon r_{\mathbf{y}}(\mathbf{x})=\bigwedge_{\mathbf{w}\in T(\mathbf{y})}\{t_{\mathbf{y}}(\mathbf{w})s_{\mathbf{w}}(\mathbf{x})\colon\mathbf{x}\in S(\mathbf{w})\}\}$. On the right hand side of the equality the image at location $\mathbf{y}\in\mathbf{Y}$ has form $\{(\mathbf{x},u_{\mathbf{y}}(\mathbf{x}))\colon u_{\mathbf{y}}(\mathbf{x})=-\bigvee_{\mathbf{w}\in T(\mathbf{y})}\{-t_{\mathbf{y}}(\mathbf{w})s_{\mathbf{w}}(\mathbf{x})\colon\mathbf{x}\in S(\mathbf{w})\}\}$. Since in general $-\{\bigvee_{\alpha\in A}\mathrm{m}_\alpha\}=\bigwedge_{\alpha\in A}\{-\mathrm{m}_\alpha\}$ for an indexing set A and a collection $\{\mathrm{m}_\alpha\}_{\alpha\in A}$, we have

$$r_{\mathbf{y}}(\mathbf{x})=\bigwedge_{\mathbf{w}\in T(\mathbf{y})}\{t_{\mathbf{y}}(\mathbf{w})s_{\mathbf{w}}(\mathbf{x})\colon\mathbf{x}\in S(\mathbf{w})\}=-\bigvee_{\mathbf{w}\in T(\mathbf{y})}\{-t_{\mathbf{y}}(\mathbf{w})s_{\mathbf{w}}(\mathbf{x})\colon\mathbf{x}\in S(\mathbf{w})\}=u_{\mathbf{y}}(\mathbf{x})$$

Q.E.D.

Lemma 4.7.2. $\mathbf{s}\textcircled{\wedge}\mathbf{t}\leq\mathbf{s}\textcircled{\vee}\mathbf{t}$ for $\mathbf{s}\in\mathbf{R}^{>0}_{\mathbf{Z}\to\mathbf{X}}$, $\mathbf{t}\in\mathbf{R}^{>0}_{\mathbf{Y}\to\mathbf{Z}}$.

Proof: We have to show that for $\mathbf{r}=\mathbf{s}\textcircled{\wedge}\mathbf{t}$ and for $\mathbf{u}=\mathbf{s}\textcircled{\vee}\mathbf{t}$ that $r_{\mathbf{y}}(\mathbf{x})\leq u_{\mathbf{y}}(\mathbf{x})$, as the configurations are equal. We have:

$r_{\mathbf{y}}(\mathbf{x})=\bigwedge_{\mathbf{w}\in T(\mathbf{y})}\{t_{\mathbf{y}}(\mathbf{w})s_{\mathbf{w}}(\mathbf{x})\colon\mathbf{x}\in S(\mathbf{w})\}$ and since all values are positive, the product is also positive, and thus we know $\bigwedge_{\mathbf{w}\in T(\mathbf{y})}\{t_{\mathbf{y}}(\mathbf{w})s_{\mathbf{w}}(\mathbf{x})\colon\mathbf{x}\in S(\mathbf{w})\}\leq\bigvee_{\mathbf{w}\in T(\mathbf{y})}\{t_{\mathbf{y}}(\mathbf{w})s_{\mathbf{w}}(\mathbf{x})\colon\mathbf{x}\in S(\mathbf{w})\}=u_{\mathbf{y}}(\mathbf{x})$.

Q.E.D.

Lemma 4.7.3. $\{ \mathbf{R}_{Y \to X}, \leq \}$ is a poset.

Proof: We use the definition $\mathbf{s} \leq \mathbf{t} \iff \mathcal{S}(\mathbf{y}) \subset \mathcal{T}(\mathbf{y})$ and $s_y \leq t_y$, $\forall\, \mathbf{y} \in \mathbf{Y}$. This obviously puts a partial order on $\mathbf{R}_{Y \to X}$, as we see that $\forall\, \mathbf{r}, \mathbf{s}, \mathbf{t} \in \mathbf{R}_{Y \to X}$,

i. $\mathbf{t} \leq \mathbf{t}$

ii. $\mathbf{t} \leq \mathbf{s}$ and $\mathbf{s} \leq \mathbf{t} \Rightarrow \mathbf{s} = \mathbf{t}$

iii. $\mathbf{r} \leq \mathbf{s}, \mathbf{s} \leq \mathbf{t} \Rightarrow \mathbf{r} \leq \mathbf{t}$

Q.E.D.

That not every pair of templates are related can be seen by the example presented in the next figure, where $\mathbf{s}, \mathbf{t} \in \mathbf{R}_{X \to X}$:

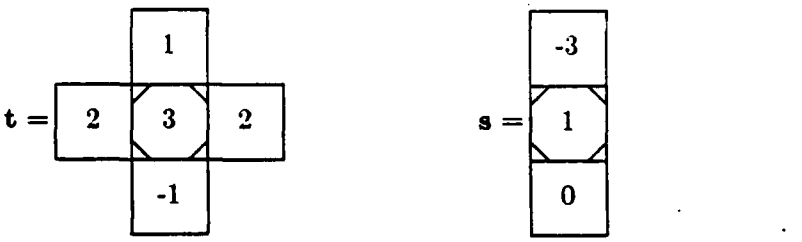


Figure 24. Two Templates Not Related Under $\leq$

Here, $\mathbf{s} \not\leq \mathbf{t}$, $\mathbf{t} \not\leq \mathbf{s}$.

Theorem 4.7.4. $(\mathbf{s} \circledcirc \mathbf{t}) \circledcirc \mathbf{r} = \mathbf{s} \circledcirc (\mathbf{t} \circledcirc \mathbf{r})$ for $\mathbf{r} \in \mathbf{R}^{>0}_{W \to Z}$, $\mathbf{t} \in \mathbf{R}^{>0}_{Z \to Y}$, $\mathbf{s} \in \mathbf{R}^{>0}_{Y \to X}$.

Proof: Let $\mathbf{u} = \mathbf{s} \circledcirc \mathbf{t} \in \mathbf{R}^{>0}_{Z \to X}$, $\mathbf{k} = (\mathbf{s} \circledcirc \mathbf{t}) \circledcirc \mathbf{r} = \mathbf{u} \circledcirc \mathbf{r} \in \mathbf{R}^{>0}_{W \to X}$, and let $\mathbf{q} = \mathbf{t} \circledcirc \mathbf{r} \in \mathbf{R}^{>0}_{W \to Y}$, $\mathbf{p} = \mathbf{s} \circledcirc (\mathbf{t} \circledcirc \mathbf{r}) = \mathbf{s} \circledcirc \mathbf{q} \in \mathbf{R}^{>0}_{W \to X}$. First we show that the configurations $\mathcal{K}$ and $\mathcal{P}$ are equal, then we show that the gray value functions k and p are equal.

$\mathcal{K}(\mathbf{w}) = \mathcal{P}(\mathbf{w})$:

$$\mathcal{K}(\mathbf{w}) = \bigcup_{\mathbf{z} \in \mathcal{R}(\mathbf{w})} \mathcal{U}(\mathbf{z}) = \bigcup_{\mathbf{z} \in \mathcal{R}(\mathbf{w})} \left(\bigcup_{\mathbf{y} \in \mathcal{T}(\mathbf{z})} \mathcal{S}(\mathbf{y}) \right), \text{ and } \mathcal{P}(\mathbf{w}) = \bigcup_{\mathbf{f} \in \mathcal{Q}(\mathbf{w})} \mathcal{S}(\mathbf{f}) = \bigcup_{\mathbf{f} \in \bigcup_{\mathbf{g} \in \mathcal{R}(\mathbf{w})} \mathcal{T}(\mathbf{g})} \mathcal{S}(\mathbf{f}).$$

Let $\mathrm{m} \in \mathcal{K}(\mathbf{w})$. Then $\exists\ \mathbf{z} \in \mathcal{R}(\mathbf{w})$ such that $\mathrm{m} \in \mathcal{U}(\mathbf{z})$. This implies that $\exists\ \mathbf{z} \in \mathcal{R}(\mathbf{w})$ and $\mathbf{y} \in \mathcal{T}(\mathbf{z})$ such that $\mathrm{m} \in \mathcal{S}(\mathbf{y})$. We need to show that $\exists\ \mathbf{f} \in \mathcal{Q}(\mathbf{w})$ such that m

$\in \mathcal{S}(\mathbf{f})$, that is, that $\exists\ \mathbf{g} \in \mathcal{R}(\mathbf{w})$ and $\mathbf{f} \in \mathcal{T}(\mathbf{g})$ such that $\mathrm{m} \in \mathcal{S}(\mathbf{f})$. So let $\mathbf{g} = \mathbf{z}$, $\mathbf{f} = \mathbf{y}$. Then $\mathbf{g} \in \mathcal{R}(\mathbf{w})$ and $\mathbf{f} \in \mathcal{T}(\mathbf{g})$ where $\mathrm{m} \in \mathcal{S}(\mathbf{f})$. Thus, $\mathcal{K}(\mathbf{w}) \subset \mathcal{A}(\mathbf{w})$. Conversely, let $\mathrm{m} \in \mathcal{A}(\mathbf{w})$. Then $\exists\ \mathbf{f} \in \mathcal{Q}(\mathbf{w})$ such that $\mathrm{m} \in \mathcal{S}(\mathbf{f})$, that is, $\exists\ \mathbf{g} \in \mathcal{R}(\mathbf{w})$ and $\mathbf{f} \in \mathcal{T}(\mathbf{g})$ such that $\mathrm{m} \in \mathcal{S}(\mathbf{f})$. Let $\mathbf{z} = \mathbf{g}$, and $\mathbf{y} = \mathbf{f}$. Then $\mathbf{z} \in \mathcal{R}(\mathbf{w})$ and $\mathbf{y} \in \mathcal{T}(\mathbf{z})$ where $\mathrm{m} \in \mathcal{S}(\mathbf{y})$. This implies that $\mathrm{m} \in \mathcal{K}(\mathbf{w})$. Thus, $\mathcal{A}(\mathbf{w}) \subset \mathcal{K}(\mathbf{w})$. If $\mathcal{K}(\mathbf{w}) = \phi$, then suppose $\mathcal{A}(\mathbf{w}) \neq \phi$. Then $\exists\ \mathrm{m} \in \mathcal{A}(\mathbf{w}) \Rightarrow$ $\exists\ \mathbf{g} \in \mathcal{R}(\mathbf{w})$ and $\mathbf{f} \in \mathcal{T}(\mathbf{g})$ such that $\mathrm{m} \in \mathcal{S}(\mathbf{f})$. Letting $\mathbf{z} = \mathbf{g}$ and $\mathbf{y} = \mathbf{f}$ we see that $\mathrm{m} \in \mathcal{K}(\mathbf{w})$, which is a contradiction. Thus, if $\mathcal{K}(\mathbf{w}) = \phi$, then $\mathcal{A}(\mathbf{w}) = \phi$, and $\mathcal{K}(\mathbf{w}) = \mathcal{A}(\mathbf{w})$.

To show that $k_{\mathbf{w}} = p_{\mathbf{w}}$, we show $k_{\mathbf{w}}(\mathbf{x}) \leq p_{\mathbf{w}}(\mathbf{x})$ and $p_{\mathbf{w}}(\mathbf{x}) \leq k_{\mathbf{w}}(\mathbf{x})$.

$k_{\mathbf{w}}(\mathbf{x}) = \bigvee\{ r_{\mathbf{w}}(\mathbf{z})\, u_{\mathbf{z}}(\mathbf{x}) : \mathbf{z} \in \mathcal{R}(\mathbf{w}) \text{ and } \mathbf{x} \in \mathcal{U}(\mathbf{z}) \}$. Since $u_{\mathbf{z}}(\mathbf{x}) =$ $\bigvee\{ t_{\mathbf{z}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{x}) : \mathbf{y} \in \mathcal{T}(\mathbf{z}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{y}) \}$, we have $k_{\mathbf{w}}(\mathbf{x}) =$

$$\bigvee\{ r_{\mathbf{w}}(\mathbf{z}) \cdot \Big[\bigvee\{ t_{\mathbf{z}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{x}) : \mathbf{y} \in \mathcal{T}(\mathbf{z}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{y}) \} \Big] : \mathbf{z} \in \mathcal{R}(\mathbf{w}) \text{ and } \mathbf{x} \in \mathcal{U}(\mathbf{z}) \} =$$

$$\bigvee \{ \bigvee \{ r_{\mathbf{w}}(\mathbf{z}) t_{\mathbf{z}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{x}) : \mathbf{y} \in \mathcal{T}(\mathbf{z}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{y}) \} : \mathbf{z} \in \mathcal{R}(\mathbf{w}) \text{ and } \mathbf{x} \in \bigcup_{\mathbf{v} \in \mathcal{T}(\mathbf{z})} \mathcal{S}(\mathbf{v}) \},$$

with this last equality following from the fact that the template weights are positive. Let D =

$$\bigcup \{ \bigcup \{ r_{\mathbf{w}}(\mathbf{z}) t_{\mathbf{z}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{x}) : \mathbf{y} \in \mathcal{T}(\mathbf{z}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{y}) \} : \mathbf{z} \in \mathcal{R}(\mathbf{w}) \text{ and } \mathbf{x} \in \bigcup_{\mathbf{v} \in \mathcal{T}(\mathbf{z})} \mathcal{S}(\mathbf{v}) \}.$$

Next, note that $p_{\mathbf{w}}(\mathbf{x}) = \bigvee\{ q_{\mathbf{w}}(\mathbf{f})\, s_{\mathbf{f}}(\mathbf{x}) : \mathbf{f} \in \mathcal{Q}(\mathbf{w}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{y}) \} =$

$$\bigvee\{ \Big[\bigvee\{ r_{\mathbf{w}}(\mathbf{n}) t_{\mathbf{n}}(\mathbf{f}) : \mathbf{n} \in \mathcal{R}(\mathbf{w}) \text{ and } \mathbf{f} \in \mathcal{T}(\mathbf{n}) \} \Big] \cdot s_{\mathbf{f}}(\mathbf{x}) : \mathbf{f} \in \bigcup_{\mathbf{g} \in \mathcal{R}(\mathbf{w})} \mathcal{T}(\mathbf{g}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{f}) \}$$

$$= \bigvee \{ \bigvee \{ r_{\mathbf{w}}(\mathbf{n}) t_{\mathbf{n}}(\mathbf{f}) s_{\mathbf{f}}(\mathbf{x}) : \mathbf{n} \in \mathcal{R}(\mathbf{w}) \text{ and } \mathbf{f} \in \mathcal{T}(\mathbf{n}) \} : \mathbf{f} \in \bigcup_{\mathbf{g} \in \mathcal{R}(\mathbf{w})} \mathcal{T}(\mathbf{g}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{f}) \}.$$

Let E =

$$\bigcup \{ \bigcup \{ r_{\mathbf{w}}(\mathbf{n}) t_{\mathbf{n}}(\mathbf{f}) s_{\mathbf{f}}(\mathbf{x}) : \mathbf{n} \in \mathcal{R}(\mathbf{w}) \text{ and } \mathbf{f} \in \mathcal{T}(\mathbf{n}) \} : \mathbf{f} \in \bigcup_{\mathbf{g} \in \mathcal{R}(\mathbf{w})} \mathcal{T}(\mathbf{g}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{f}) \}.$$

An element of E is of form $r_{\mathbf{w}}(\mathbf{n}) t_{\mathbf{n}}(\mathbf{f}) s_{\mathbf{f}}(\mathbf{x})$, where $\mathbf{n} \in \mathcal{R}(\mathbf{w})$, $\mathbf{f} \in \mathcal{T}(\mathbf{n})$, and $\mathbf{x} \in \mathcal{S}(\mathbf{f})$. We will now show that D $\subset$ E. To this end, choose $\mathbf{z} \in \mathcal{R}(\mathbf{w})$ such that $\mathbf{x} \in \mathcal{U}(\mathbf{z})$. Therefore, $\exists\ \mathbf{y} \in \mathcal{T}(\mathbf{z})$ such that $\mathbf{x} \in \mathcal{S}(\mathbf{y})$. Thus, $r_{\mathbf{w}}(\mathbf{z}) t_{\mathbf{z}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{x}) \in$ D. To show that $r_{\mathbf{w}}(\mathbf{z}) t_{\mathbf{z}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{x}) \in$ E, we must show that for $\mathbf{n} = \mathbf{z}$, $\exists\ \mathbf{f} \in \mathcal{T}(\mathbf{n})$ such that $\mathbf{f} = \mathbf{y}$, and that $\mathbf{x} \in \mathcal{S}(\mathbf{f})$. But we know this is true, by choosing $\mathbf{f} = \mathbf{y}$, as $\mathbf{y} \in \mathcal{T}(\mathbf{z})$ and $\mathbf{x} \in \mathcal{S}(\mathbf{y})$. Thus, $r_{\mathbf{w}}(\mathbf{z}) t_{\mathbf{z}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{x}) = r_{\mathbf{w}}(\mathbf{n}) t_{\mathbf{n}}(\mathbf{f}) s_{\mathbf{f}}(\mathbf{x}) \in$ E. Therefore, $k_{\mathbf{w}}(\mathbf{x}) \leq p_{\mathbf{w}}(\mathbf{x})$. Conversely, for $r_{\mathbf{w}}(\mathbf{n}) t_{\mathbf{n}}(\mathbf{f}) s_{\mathbf{f}}(\mathbf{x}) \in$ E, where $\mathbf{f} \in \mathcal{T}(\mathbf{n})$ for some

$\mathbf{n} \in \mathcal{R}(\mathbf{w})$ and $\mathbf{x} \in \mathcal{S}(\mathbf{f})$, we must show that $r_{\mathbf{w}}(\mathbf{n})t_{\mathbf{n}}(\mathbf{f})s_{\mathbf{f}}(\mathbf{x})$ is of form $r_{\mathbf{w}}(\mathbf{z})t_{\mathbf{z}}(\mathbf{y})s_{\mathbf{y}}(\mathbf{x})$, for some $\mathbf{z} \in \mathcal{R}(\mathbf{w})$, where $\mathbf{y} \in \mathcal{T}(\mathbf{z})$ and $\mathbf{x} \in \mathcal{S}(\mathbf{y})$. Here, we choose $\mathbf{z} = \mathbf{n}$, and $\mathbf{y} = \mathbf{f}$, and we are done.

Lastly, we note that if $\mathcal{K}(\mathbf{w}) = \phi$, then $k_{\mathbf{w}} = \mathbf{0}$. Since in this case $\mathcal{F}(\mathbf{w}) = \phi$, we have $p_{\mathbf{w}} = \mathbf{0}$ also.

Q.E.D.

Corollary 4.7.5. $(\mathbf{s} \textcircled{\scriptsize A} \mathbf{t}) \textcircled{\scriptsize A} \mathbf{r} = \mathbf{s} \textcircled{\scriptsize A} (\mathbf{t} \textcircled{\scriptsize A} \mathbf{r})$ for $\mathbf{r} \in \mathbf{R}^{>0}_{\mathbf{W} \to \mathbf{Z}}$, $\mathbf{t} \in \mathbf{R}^{>0}_{\mathbf{Z} \to \mathbf{Y}}$, $\mathbf{s} \in \mathbf{R}^{>0}_{\mathbf{Y} \to \mathbf{X}}$.

Proof: The proof is the same as for Theorem 4.7.4, but when taking the minimum of a set, we note that since the weights of the templates are strictly positive on their configurations, the minimum is never zero.

Note that if one of $\mathbf{s}$, $\mathbf{t}$, or $\mathbf{r}$ is not positive, then Theorem 4.7.4 is not necessarily true. For example, let $\mathbf{s}$, $\mathbf{t}$, $\mathbf{r} \in \mathbf{R}_{\mathbf{X} \to \mathbf{X}}$, as in Figure 25 below.

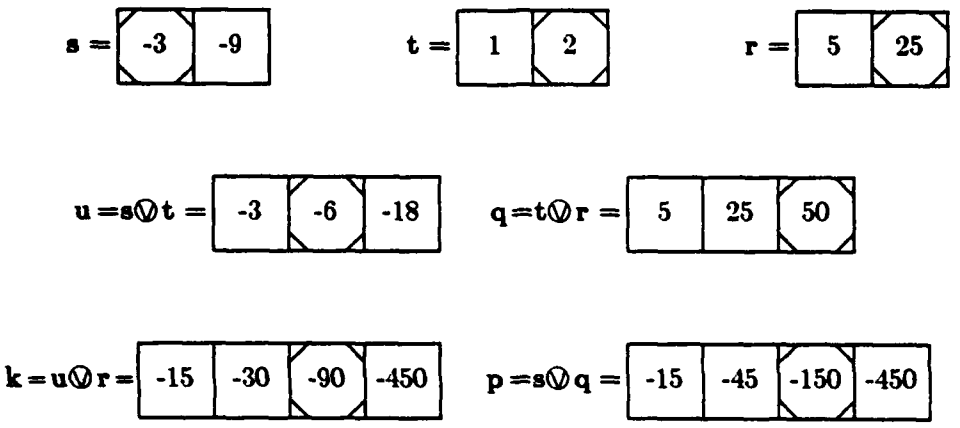


Figure 25. $(\mathbf{s} \circledcirc \mathbf{t}) \circledcirc \mathbf{r} \neq \mathbf{s} \circledcirc (\mathbf{t} \circledcirc \mathbf{r})$

Thus, obviously, $\mathbf{k} \neq \mathbf{p}$.

We also remark that since the template weights are all positive in Theorem 4.7.4 and Corollary 4.7.5, that an abbreviated definition for the gray value function of $\mathbf{r} = \mathbf{s} \circledcirc \mathbf{t}$ can be used in place of its usual definition in proving these results, namely

$$r_y(x) = \bigvee \{ t_y(z) s_z(x) : z \in T(y) \},$$

and equivalently for $\circledcirc$. This is because if, for some $z \in T(y)$ there is an $x \notin S(z)$, then $s_z(x) = 0$, which makes the product $t_y(z)s_z(x) = 0$. Since all such products are positive anyway, the computation of $r_y(x)$ is the same using either definition.

Recalling that a semigroup is a set having an associative operation, we can say that

Corollary 4.7.6. $\{ \mathbf{R}^{>0}_{X \to X}, \circledcirc \}$ is a semi-group.

Corollary 4.7.7. $\{ \mathbf{R}^{>0}_{X \to X}, \circledcirc \}$ is a semi-group.

If $\{ \mathbf{R}^{>0}_{Y \to X}, \vee, \wedge \}$ were a lattice, we could obtain a richer, more useful structure, as many properties of lattices are well-known. However, only three of the four properties necessary (see Reference 28, for example) hold. The proofs of these are trivial:

i. $\mathbf{s} \wedge \mathbf{s} = \mathbf{s}, \mathbf{s} \vee \mathbf{s} = \mathbf{s}$

ii. $\mathbf{s} \wedge \mathbf{t} = \mathbf{t} \wedge \mathbf{s}, \mathbf{s} \vee \mathbf{t} = \mathbf{t} \vee \mathbf{s}$

iii. $(\mathbf{s} \wedge \mathbf{t}) \wedge \mathbf{u} = \mathbf{s} \wedge (\mathbf{t} \wedge \mathbf{u}), (\mathbf{s} \vee \mathbf{t}) \vee \mathbf{u} = \mathbf{s} \vee (\mathbf{t} \vee \mathbf{u}),$

It is the fourth one that does not hold,

iv. $\mathbf{s} \wedge (\mathbf{s} \vee \mathbf{t}) = \mathbf{s} \vee (\mathbf{s} \wedge \mathbf{t}) = \mathbf{s}$

since by definition of $\mathbf{s} \vee \mathbf{t}$, the resulting configuration at $\mathbf{y}$ is $S(\mathbf{y}) \cup T(\mathbf{y})$, which is not necessarily the same as $S(\mathbf{y})$. A counterexample is easily constructed. This is presented in Figure 26, below.

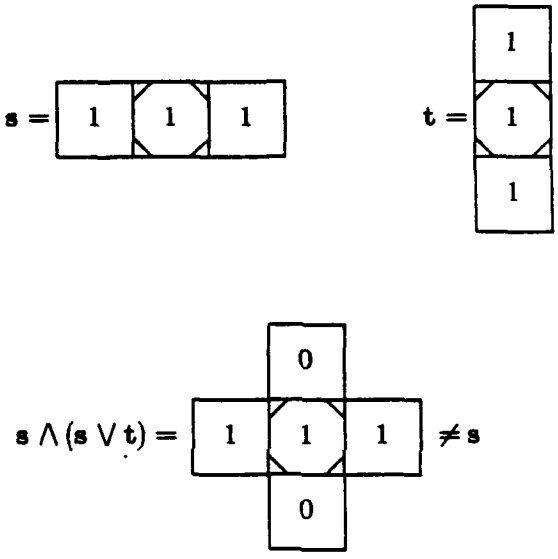


Figure 26. $\mathbf{s} \wedge (\mathbf{s} \vee \mathbf{t}) \neq \mathbf{s}$

That $\mathbf{s} \wedge (\mathbf{s} \vee \mathbf{t}) \neq \mathbf{s}$ follows from the definition of equality of templates.

This makes the structures $(\mathbf{R}^{>0}_{\mathbf{Y}\to\mathbf{X}}, \vee, \leq)$ and $(\mathbf{R}^{>0}_{\mathbf{Y}\to\mathbf{X}}, \wedge, \leq)$ semi-lattices, not lattices. To obtain a richer structure, we use the equivalence relation between templates as defined by P. Gader 6, that is,

$$\mathbf{s} \sim \mathbf{t} \text{ if and only if } t_{\mathbf{y}} = s_{\mathbf{y}} \; \forall \, \mathbf{y} \in \mathbf{Y}.$$

This equivalence relation is well defined under $\circledcirc$ for templates and images. Thus for $\mathbf{s}_1 \sim \mathbf{s}_2$, $\mathbf{t}_1 \sim \mathbf{t}_2$, when all templates are elements of $\mathbf{R}^{>0}_{\mathbf{X}\to\mathbf{X}}$, we have

i. $[\mathbf{s}_1 \circledcirc \mathbf{t}_1] = [\mathbf{s}_2 \circledcirc \mathbf{t}_2] = [\mathbf{s}_1] \circledcirc [\mathbf{t}_1]$

ii. $[\mathbf{s}_1 \vee \mathbf{t}_1] = [\mathbf{s}_2 \vee \mathbf{t}_2] = [\mathbf{s}_1] \vee [\mathbf{t}_1]$

iii. $\mathbf{a} \circledcirc \mathbf{s}_1 = \mathbf{a} \circledcirc \mathbf{s}_2$, if $\mathbf{a} \geq 0$, and $\mathbf{s}_1 > 0$.

We denote this set of equivalence classes by $E(\mathbf{R}^{>0}_{X \to X})$. Under this equivalence relation, equality of two templates reduces to equality of the two gray value functions only.

Since $\sim$ is well-defined under $\circledcirc$, associativity is preserved for equivalence classes of templates, that is, $([\mathbf{s}] \circledcirc [\mathbf{t}]) \circledcirc [\mathbf{r}] = [\mathbf{s}] \circledcirc ([\mathbf{t}] \circledcirc [\mathbf{r}])$. Thus we have

Theorem 4.7.8. $\{ E(\mathbf{R}^{>0}_{X \to X}), \circledcirc \}$ is a semi-group.

Corollary 4.7.9. $\{ E(\mathbf{R}^{>0}_{X \to X}), \circledast \}$ is a semi-group.

Lemma 4.7.10. Let $[\mathbf{s}], [\mathbf{t}] \in E(\mathbf{R}^{>0}_{Y \to X})$. Then $[\mathbf{s}] \wedge ([\mathbf{s}] \vee [\mathbf{t}]) = [\mathbf{s}] \vee ([\mathbf{s}] \wedge [\mathbf{t}]) = [\mathbf{s}]$.

Proof: Since $[\mathbf{s}] = \{ \mathbf{u} \in \mathbf{R}^{>0}_{Y \to X} : \mathbf{u} \sim \mathbf{s} \} = \{ \mathbf{u} \in \mathbf{R}^{>0}_{Y \to X} : u_y = s_y \; \forall \, \mathbf{y} \in \mathbf{Y} \}$, we need only to show that the gray value functions are equal, as mentioned above. Let $\mathbf{r} = \mathbf{s} \vee \mathbf{t}$, and $\mathbf{u} = \mathbf{s} \wedge \mathbf{t}$, $\mathbf{p} = \mathbf{s} \wedge \mathbf{r}$, and $\mathbf{q} = \mathbf{s} \vee \mathbf{u}$. Then $p_y(\mathbf{x}) = s_y(\mathbf{x}) \wedge \{ s_y(\mathbf{x}) \vee t_y(\mathbf{x}) \}$, and $q_y(\mathbf{x}) = s_y(\mathbf{x}) \vee \{ s_y(\mathbf{x}) \wedge t_y(\mathbf{x}) \}$. Since $\mathbf{R}$ itself is a lattice, we see immediately that $p_y(\mathbf{x}) = q_y(\mathbf{x}) = s_y(\mathbf{x})$.

Q.E.D.

This helps to prove that

Theorem 4.7.11. $\{ E(\mathbf{R}^{>0}_{Y \to X}), \vee \}$ is a lattice.

Proof: This follows from properties i, ii, iii, the equivalence relation under $\circledcirc$, and Lemma 4.7.10.

Corollary 4.7.12. $\{ E(\mathbf{R}^{>0}_{Y \to X}), \wedge \}$ is a lattice.

Much is known about lattices. Some properties immediately carry over to $\{ E(\mathbf{R}^{>0}_{Y \to X}), \vee \}$. For example, we can say that $\{E(\mathbf{R}^{>0}_{Y \to X}), \vee \}$ is a distributive lattice, since $\mathbf{R}$ is itself a distributive lattice. This follows trivially from the fact that the gray value function of $\mathbf{s} \vee \mathbf{t}$ is computed pointwise, thus inducing the distributive lattice property of the real numbers on $E(\mathbf{R}^{>0}_{Y \to X})$. Also, any distributive lattice is modular. Thus, we can apply the full theory of distributive and modular lattices to $E(\mathbf{R}^{>0}_{Y \to X})$. As an example, in any distributive lattice $[\mathbf{s}] \wedge ([\mathbf{t}] \vee [\mathbf{r}]) = ([\mathbf{s}] \wedge [\mathbf{t}]) \vee ([\mathbf{s}] \wedge [\mathbf{r}])$. There are too many theorems to state them all here, but Reference 28 can be consulted for more results.

A multiplicative semilattice or an m-semilattice is a semilattice M under $\vee$ with a multiplication such that $\forall$ a,b,c $\in$ M,

$$a(b \vee c) = ab \vee ac \quad \text{and} \quad (a \vee b)c = ac \vee c$$

If M is a lattice with a multiplication and the above property is true also, then M is called an m-lattice or an l-groupoid. An m-lattice which is a semigroup (monoid) under multiplication is called an l-semigroup (respectively an l-monoid, short for lattice-ordered monoid), Reference 28.

A multiplicative poset or an m-poset is a poset M with a binary multiplication which satisfies

$$a \leq b \quad \text{implies} \quad xa \leq xb \quad \text{and} \quad ax \leq bx$$

for all a, b, $x \in M$. It follows easily (using the above property of an m-semilattice) that any m-semilattice is an m-poset. We will show that some of these substructures exist in the Image Algebra.

Theorem 4.7.13. $\mathbf{s} \circledcirc (\mathbf{t} \vee \mathbf{r}) = (\mathbf{s} \circledcirc \mathbf{t}) \vee (\mathbf{s} \circledcirc \mathbf{r})$ if $\mathbf{t}, \mathbf{r} \in \mathbf{R}^{>0}_{Y \to Z}$, and $\mathbf{s} \in \mathbf{R}^{>0}_{Z \to X}$.

Proof: Let $\mathbf{u} = \mathbf{t} \vee \mathbf{r}$, $\mathbf{k} = \mathbf{s} \circledcirc (\mathbf{t} \vee \mathbf{r}) = \mathbf{s} \circledcirc \mathbf{u}$, $\mathbf{p} = \mathbf{s} \circledcirc \mathbf{t}$, $\mathbf{q} = \mathbf{s} \circledcirc \mathbf{r}$, and $\mathbf{l} = (\mathbf{s} \circledcirc \mathbf{t}) \vee (\mathbf{s} \circledcirc \mathbf{r}) = \mathbf{p} \vee \mathbf{q}$.

$$\text{Then } \mathcal{K}(\mathbf{y}) = \bigcup_{\mathbf{z} \in \mathcal{T}(\mathbf{y}) \cup \mathcal{R}(\mathbf{y})} \mathcal{S}(\mathbf{z}) = \bigcup_{\mathbf{z} \in \mathcal{T}(\mathbf{y})} \mathcal{S}(\mathbf{z}) \cup \bigcup_{\mathbf{m} \in \mathcal{R}(\mathbf{y})} \mathcal{S}(\mathbf{m}).$$

Since $\mathcal{L}(\mathbf{y}) = \mathcal{P}(\mathbf{y}) \cup \mathcal{Q}(\mathbf{y}) = \bigcup_{\mathbf{z} \in \mathcal{T}(\mathbf{y})} \mathcal{S}(\mathbf{z}) \cup \bigcup_{\mathbf{m} \in \mathcal{R}(\mathbf{y})} \mathcal{S}(\mathbf{m})$, we see immediately that $\mathcal{K}(\mathbf{y}) = \mathcal{L}(\mathbf{y})$. As for the gray values $k_{\mathbf{y}}(\mathbf{x})$ and $l_{\mathbf{y}}(\mathbf{x})$:

$$k_{\mathbf{y}}(\mathbf{x}) = \vee \{ u_{\mathbf{y}}(\mathbf{z}) a_{\mathbf{z}}(\mathbf{x}) : \mathbf{z} \in \mathcal{U}(\mathbf{y}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{z}) \}$$

$$= \vee \{ [t_{\mathbf{y}}(\mathbf{z}) \vee r_{\mathbf{y}}(\mathbf{z})] a_{\mathbf{z}}(\mathbf{x}) : \mathbf{z} \in \mathcal{T}(\mathbf{y}) \cup \mathcal{R}(\mathbf{y}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{z}) \}$$

$$= \vee \{ t_{\mathbf{y}}(\mathbf{z}) a_{\mathbf{z}}(\mathbf{x}) \vee r_{\mathbf{y}}(\mathbf{z}) a_{\mathbf{z}}(\mathbf{x}) : \mathbf{z} \in \mathcal{T}(\mathbf{y}) \cup \mathcal{R}(\mathbf{y}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{z}) \},$$

with this last equality following from the fact that the template weights are positive. Continuing,

$$k_{\mathbf{y}}(\mathbf{x}) = \vee \{ t_{\mathbf{y}}(\mathbf{z}) a_{\mathbf{z}}(\mathbf{x}) : \mathbf{z} \in \mathcal{T}(\mathbf{y}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{z}) \} \vee \vee \{ r_{\mathbf{y}}(\mathbf{z}) a_{\mathbf{z}}(\mathbf{x}) : \mathbf{z} \in \mathcal{R}(\mathbf{y}) \text{ and } \mathbf{x} \in \mathcal{S}(\mathbf{z}) \},$$

as $\mathbf{r}$, $\mathbf{s}$, and $\mathbf{t}$ are positive and have zero weight value outside of their respective configurations. This is exactly $l_{\mathbf{y}}(\mathbf{x})$. Q.E.D.

Corollary 4.7.14 $(\mathbf{t} \vee \mathbf{r}) \circledcirc \mathbf{s} = (\mathbf{t} \circledcirc \mathbf{s}) \vee (\mathbf{r} \circledcirc \mathbf{s})$, if $\mathbf{t}, \mathbf{r} \in \mathbf{R}^{>0}_{Z \to X}$, and $\mathbf{s} \in \mathbf{R}^{>0}_{Y \to Z}$.

Proof: We omit this, as it is very similar to the proof of Theorem 4.7.13.

Corollary 4.7.15. $\mathbf{s} \oslash (\mathbf{t} \wedge \mathbf{r}) = (\mathbf{s} \oslash \mathbf{t}) \wedge (\mathbf{s} \oslash \mathbf{r})$ if $\mathbf{t}, \mathbf{r} \in \mathbf{R}^{\geq 0}_{\mathbf{Z} \to \mathbf{X}}$, and $\mathbf{s} \in \mathbf{R}^{\geq 0}_{\mathbf{Y} \to \mathbf{Z}}$.

Corollary 4.7.16 $(\mathbf{t} \wedge \mathbf{r}) \oslash \mathbf{s} = (\mathbf{t} \oslash \mathbf{s}) \wedge (\mathbf{r} \oslash \mathbf{s})$, if $\mathbf{t}, \mathbf{r} \in \mathbf{R}^{\geq 0}_{\mathbf{Z} \to \mathbf{X}}$, and $\mathbf{s} \in \mathbf{R}^{\geq 0}_{\mathbf{Y} \to \mathbf{Z}}$.

At this point, a number of results follow immediately as a direct consequence of the previous properties. For example, as a consequence of Theorems 4.7.8, 4.7.11, 4.7.13, and Corollary 4.7.14, we have

Theorem 4.7.17. $\{ E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \vee, \otimes \}$ *is an* m-lattice.

and the dual result:

Corollary 4.7.18. $\{ E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \wedge, \oslash \}$ is an m-lattice.

Lemma 4.7.19. Let $[\mathbf{s}],[\mathbf{t}] \in E(\mathbf{R}^{\geq 0}_{\mathbf{Z} \to \mathbf{Y}})$ such that $[\mathbf{s}] \leq [\mathbf{t}]$, and let $[\mathbf{r}] \in E(\mathbf{R}^{\geq 0}_{\mathbf{W} \to \mathbf{Z}})$, $[\mathbf{u}] \in E(\mathbf{R}^{\geq 0}_{\mathbf{Y} \to \mathbf{X}})$. Then $[\mathbf{u}] \otimes [\mathbf{s}] \otimes [\mathbf{r}] \leq [\mathbf{u}] \otimes [\mathbf{t}] \otimes [\mathbf{r}]$.

Corollary 4.7.20. Let $[\mathbf{s}],[\mathbf{t}] \in E(\mathbf{R}^{\geq 0}_{\mathbf{Z} \to \mathbf{Y}})$ such that $[\mathbf{s}] \leq [\mathbf{t}]$, and let $[\mathbf{r}] \in E(\mathbf{R}^{\geq 0}_{\mathbf{W} \to \mathbf{Z}})$, $[\mathbf{u}] \in E(\mathbf{R}^{\geq 0}_{\mathbf{Y} \to \mathbf{X}})$. Then $[\mathbf{u}] \oslash [\mathbf{s}] \oslash [\mathbf{r}] \leq [\mathbf{u}] \oslash [\mathbf{t}] \oslash [\mathbf{r}]$.

Corollary 4.7.21. $([\mathbf{s}] \wedge [\mathbf{t}]) \otimes ([\mathbf{s}] \vee [\mathbf{t}]) \leq ([\mathbf{t}] \otimes [\mathbf{s}]) \vee ([\mathbf{s}] \otimes [\mathbf{t}])$ for $[\mathbf{s}], [\mathbf{t}] \in E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}})$.

Theorem 4.7.22. $\{ E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \vee, \otimes \}$ is a lattice-ordered semi-group (an l-semigroup).

Corollary 4.7.23. $\{ E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \wedge, \oslash \}$ is a lattice-ordered semi-group (an l-semigroup).

Recalling that a monoid is a set M which has an associative binary operation and an identity element $I \in M$ such that $I*x = x*I = x \; \forall \; x \in M$, it is trivial to show that

Lemma 4.7.24. $\{E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \otimes, [I]\}$ is a monoid where

$$[I] = \{\mathbf{t} \in \mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}} : t_{\mathbf{x}}(\mathbf{x}) = 1, \text{ and } t_{\mathbf{x}}(\mathbf{y}) = 0 \text{ if } \mathbf{y} \neq \mathbf{x}\}.$$

Lemma 4.7.25. $\{E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \oslash, [I]\}$ is a monoid where I is as in Lemma 4.7.24.

Theorem 4.7.17 and Lemma 4.7.24 prove that

Theorem 4.7.26. $\{ E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \otimes, \vee, [I] \}$ is an l-monoid,

and of course its dual:

Corollary 4.7.27. $\{ E(\mathbf{R}^{\geq 0}_{\mathbf{X} \to \mathbf{X}}), \oslash, \wedge, [I] \}$ is an l-monoid.

We now view the set $\mathbf{X}$ of image coordinates as a rectangular parallelepiped. More specifically, let $\{n_1, n_2, ..., n_k\}$ be a set of positive integers and $\mathbf{X} =$ $\{(x_1, ..., x_k) : 0 \leq x_i \leq n_i, x_i \in \mathbf{Z}^k, i=1, ..., k\}$.

Definition. If $\mathbf{z} = (z_1, \cdots, z_k) \in \mathbf{Z}^k$, then the congruence class of $\mathbf{z}$ mod $\mathbf{X}$, denoted by $<\mathbf{z}>$, is defined as $<\mathbf{z}> = (z_1 \bmod(n_1+1), ..., z_k \bmod(n_k+1))$. If $\mathbf{Y} \subset \mathbf{Z}^k$ then the congruence class of $\mathbf{Y}$ mod $\mathbf{X}$ is defined as $<\mathbf{Y}> = \{<\mathbf{y}> : \mathbf{y} \in \mathbf{Y}\}$. We observe that $<\mathbf{z}> \in \mathbf{X}$, $<\mathbf{Y}> \subset \mathbf{X}$, $<\mathbf{Z}^k> = \mathbf{X} = <\mathbf{X}>$, and if $\mathbf{x} \in \mathbf{X}$, then $<\mathbf{x}> = \mathbf{x}$.

Lemma 4.7.28. If $\mathbf{t}$ is translation invariant, then $t'_x(\mathbf{y}) = t_y(\mathbf{x})$.

Proof: $t'_x(\mathbf{y}) = t_x(<2\mathbf{x} - \mathbf{y}>) = t_{<\mathbf{x}+\mathbf{z}>}(<2\mathbf{x} - \mathbf{y} + \mathbf{z}>) = t_y(\mathbf{x})$, where $\mathbf{z} = \mathbf{y} - \mathbf{x}$.

Q.E.D.

Lemma 4.7.29. $(\mathbf{s} \circledcirc \mathbf{t})_y = t_y \circledcirc \mathbf{s}'$, $\mathbf{s} \in \mathbf{R}^{>0}_{X \to X}$, $\mathbf{t} \in \mathbf{R}^{>0}_{Y \to X}$ and $\mathbf{s}$ translation invariant.

Proof: $(\mathbf{s} \circledcirc \mathbf{t})_y = \{(\mathbf{z}, r_y(\mathbf{z})) : r_y(\mathbf{z}) = \bigvee\{ t_y(\mathbf{x}) s_x(\mathbf{z}) : \mathbf{x} \in \mathcal{T}(\mathbf{y}) \text{ and } \mathbf{z} \in \mathcal{S}(\mathbf{x})\} \}$ while

$t_y \circledcirc \mathbf{s}' = \{(\mathbf{z}, \mathbf{c}(\mathbf{z})) : \mathbf{c}(\mathbf{z}) = \bigvee\{ t_y(\mathbf{w}) s'_z(\mathbf{w}) : \mathbf{w} \in \mathcal{S}'(\mathbf{z})\} \}$. We have

$\mathbf{c}(\mathbf{z}) = \bigvee\{ t_y(\mathbf{w}) s'_z(\mathbf{w}) : \mathbf{w} \in \mathcal{S}'(\mathbf{z})\} = \bigvee\{ t_y(\mathbf{w}) s'_z(\mathbf{w}) : \mathbf{w} \in \mathcal{T}(\mathbf{y})\}$

since $t_y(\mathbf{w}) = 0$ if $\mathbf{w} \notin \mathcal{T}(\mathbf{y})$ and s_y, $t_y \geq 0$.

Continuing,

$\mathbf{c}(\mathbf{z}) = \bigvee\{ t_y(\mathbf{w}) s_w(\mathbf{z}) : \mathbf{w} \in \mathcal{T}(\mathbf{y})\} = \bigvee\{ t_y(\mathbf{w}) s_w(\mathbf{z}) : \mathbf{w} \in \mathcal{T}(\mathbf{y}) \text{ and } \mathbf{z} \in \mathcal{S}(\mathbf{w})\}$ since $s_w(\mathbf{z}) = 0$ if $\mathbf{z} \notin \mathcal{S}(\mathbf{w})$. This is exactly $r_y(\mathbf{z})$.

Q.E.D.

Corollary 4.7.30. $(\mathbf{s} \circledcirc \mathbf{t})_y = t_y \circledcirc \mathbf{s}'$, $\mathbf{s} \in \mathbf{R}^{>0}_{X \to X}$, $\mathbf{t} \in \mathbf{R}^{>0}_{Y \to X}$ and $\mathbf{s}$ translation invariant.

The next two results, Theorems 4.7.31 and 4.7.32, have been proven in Reference 29. We shall use them in proving Lemma 4.7.33.

Theorem 4.7.31. If $\mathbf{t}$ is circulant, then $\mathbf{t}$ is translation invariant.

Theorem 4.7.32. If $\mathcal{S}$ and $\mathcal{T}$ are circulant, then

$$\bigcup_{\mathbf{y} \in \mathcal{T}(\mathbf{x})} \mathcal{S}(\mathbf{y}) = \bigcup_{\mathbf{z} \in \mathcal{S}(\mathbf{x})} \mathcal{T}(\mathbf{z}) \text{ for all } \mathbf{x} \in \mathbf{X}.$$

Lemma 4.7.33. $\mathbf{s} \circledcirc \mathbf{t} = \mathbf{t} \circledcirc \mathbf{s}$, for $\mathbf{s}$, $\mathbf{t} \in \mathbf{R}^{>0}_{X \to X}$ and $\mathbf{s}$,$\mathbf{t}$ circulant.

Proof: By Theorem 4.7.32, the configurations are the same. Thus all we need to show is $(\mathbf{s} \circledcirc \mathbf{t})_y = (\mathbf{t} \circledcirc \mathbf{s})_y$. We know that $(\mathbf{s} \circledcirc \mathbf{t})_y = t_y \circledcirc \mathbf{s}'$, as by Theorem 4.7.31 $\mathbf{t}$ is translation invariant, and thus the gray value at location $\mathbf{x} \in \mathbf{X}$ of $t_y \circledcirc \mathbf{s}'$ is of form $\mathbf{c}(\mathbf{x})$

$= \bigvee\{ t_y(\mathbf{w}) s'_x(\mathbf{w}) : \mathbf{w} \in \mathcal{S}'(\mathbf{x})\}$

$= \bigvee\{ t'_{\mathbf{w}}(\mathbf{y}) s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \}$ by Lemma 4.7.28

$= \bigvee\{ t'_{<\mathbf{w}+\mathbf{x}-\mathbf{w}>}(<\mathbf{y}+\mathbf{x}-\mathbf{w}>) s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \}$, by the invariance of $\mathbf{t}$

$= \bigvee\{ t'_{\mathbf{x}}(<\mathbf{y}+\mathbf{x}-\mathbf{w}>) s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \}$

$= \bigvee\{ t'_{\mathbf{x}}(\mathbf{p}) s_{<\mathbf{y}+\mathbf{x}-\mathbf{p}>}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{y}+\mathbf{x}-\mathbf{p}) \}$, letting $\mathbf{p} = \mathbf{y} + \mathbf{x} - \mathbf{w}$;

$= \bigvee\{ t'_{\mathbf{x}}(\mathbf{p}) s_{<\mathbf{y}+\mathbf{x}-\mathbf{p}+(\mathbf{p}-\mathbf{x})>}(<\mathbf{x}+\mathbf{p}-\mathbf{x}>) : \mathbf{p} \in S(\mathbf{y}) \}$ by the invariance of $\mathbf{s}$

$= \bigvee\{ t'_{\mathbf{x}}(\mathbf{p}) s_{\mathbf{y}}(\mathbf{p}) : \mathbf{p} \in S(\mathbf{y}) \}$

$= \bigvee\{ t'_{\mathbf{x}}(\mathbf{p}) s_{\mathbf{y}}(\mathbf{p}) : \mathbf{p} \in T'(\mathbf{x}) \}$ as the product is zero unless $\mathbf{p} \in T'(\mathbf{x})$,

which is the gray value at location x of $s_{\mathbf{y}} \textcircled{\vee} t' = (t \textcircled{\vee} s)_{\mathbf{y}}$.

Q.E.D.

Corollary 4.7.34. $s \textcircled{\wedge} t = t \textcircled{\wedge} s$, for $\mathbf{s}, \mathbf{t} \in \mathbf{R}^{\geq 0}_{X \to X}$ and $\mathbf{s}$,$\mathbf{t}$ circulant.

Theorem 4.7.35. ($C(\mathbf{R}^{\geq 0}_{X \to X})$, $\textcircled{\vee}$, $\vee$, [I]) is a commutative l-monoid, where $C(\mathbf{R}^{\geq 0}_{X \to X})$ is the set of equivalence classes containing circulant templates.

Corollary 4.7.36. ($C(\mathbf{R}^{\geq 0}_{X \to X})$, $\textcircled{\wedge}$, $\wedge$, [I]) is a commutative l-monoid.

Several comments are in order here. First, unlike the linear algebra structure of the Image Algebra which was shown to be an isomorphism, we have no isomorphism between the lattice structure and subalgebras of the Image Algebra. It is a subtle but distinct difference. Second, the structures in this section do not involve images, that is, elements of $\mathbf{R}^{X}$. However, some useful properties of images and templates with respect to $\textcircled{\vee}$ have been identified, and these are present in the next section.

b. Operations Between Images and Templates under $\textcircled{\vee}$

This section presents properties involving images and templates under $\textcircled{\vee}$, $\vee$, and $+$. There is no well-known structure to which these identities adhere.

Lemma 4.7.37. $(\mathbf{a} \textcircled{\vee} \mathbf{s}) \textcircled{\vee} \mathbf{t} = \mathbf{a} \textcircled{\vee} (\mathbf{s} \textcircled{\vee} \mathbf{t})$, for $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$, $\mathbf{s} \in \mathbf{R}^{\geq 0}_{Z \to X}$, and $\mathbf{t} \in \mathbf{R}^{\geq 0}_{Y \to Z}$.

Proof: Let $\mathbf{c} = (\mathbf{a} \textcircled{\vee} \mathbf{s}) \textcircled{\vee} \mathbf{t}$, $\mathbf{r} = \mathbf{s} \textcircled{\vee} \mathbf{t}$, and $\mathbf{d} = \mathbf{a} \textcircled{\vee} (\mathbf{s} \textcircled{\vee} \mathbf{t}) = \mathbf{a} \textcircled{\vee} \mathbf{r}$. Since both $\mathbf{c}$ and $\mathbf{d}$ are images on $\mathbf{Y}$, we need only to show that for an arbitrary $\mathbf{y} \in \mathbf{Y}$, $\mathbf{c}(\mathbf{y}) = \mathbf{d}(\mathbf{y})$. Note that $\mathbf{c}(\mathbf{y}) = \bigvee_{\mathbf{z} \in T(\mathbf{y})} \left(\bigvee_{\mathbf{x} \in S(\mathbf{z})} \mathbf{a}(\mathbf{x}) s_{\mathbf{z}}(\mathbf{x}) \right) t_{\mathbf{y}}(\mathbf{z}) = \bigvee_{\mathbf{z} \in T(\mathbf{y})} \left(\bigvee_{\mathbf{x} \in S(\mathbf{z})} \mathbf{a}(\mathbf{x}) s_{\mathbf{z}}(\mathbf{x}) t_{\mathbf{y}}(\mathbf{z}) \right)$. Also, $\mathbf{d}(\mathbf{y}) = \bigvee_{\mathbf{p} \in R(\mathbf{y})} \mathbf{a}(\mathbf{p}) r_{\mathbf{y}}(\mathbf{p}) = \bigvee_{\mathbf{p} \in \bigcup_{\mathbf{m} \in T(\mathbf{y})} S(\mathbf{m})} \left[\bigvee\{ \mathbf{a}(\mathbf{p}) t_{\mathbf{y}}(\mathbf{n}) s_{\mathbf{n}}(\mathbf{p}) : \mathbf{n} \in T(\mathbf{y}) \text{ and } \mathbf{p} \in S(\mathbf{n}) \right]$.

Let $Q = \bigcup_{z \in T(y)} \left(\bigcup_{x \in S(z)} a(x) s_z(x) t_y(z) \right)$, and $W = \bigcup_{p \in \bigcup_{m \in T(y)} S(m)} \left[\cup \{ a(p) t_y(n) s_n(p) : n \in T(y) \text{ and } p \in S(n) \} \right]$. An element of Q is of form $a(x) t_y(z) s_z(x)$, where $z \in T(y)$ and $x \in S(z)$. To show that this element is also in W, we have to show that for $n = z \in T(y)$ that $\exists\ p \in S(n)$ such that $p = x$. Note in enumerating the elements in W, that once $p \in \bigcup_{m \in T(y)} S(m)$ is chosen, then all $m \in T(y)$ such that $p \in S(m)$ can be listed and the inner union in W is over all such m, that is: for $p \in \bigcup_{m \in T(y)} S(m)$, let $M = \{m_1, \ldots, m_k\}$ be the set of all $m \in T(y)$ such that $p \in S(m)$. Then $\cup \{ a(p) t_y(n) s_n(p) : n \in T(y) \text{ and } p \in S(n) \} = \bigcup_{i=1}^{k} a(p) t_y(m_i) s_{m_i}(p)$. Thus, since $a(x) t_y(z) s_z(x) \in Q$ where $z \in T(y)$ and $x \in S(z)$, set $p = x$. Hence $\exists\ m_i$ such that $m_i = z$, as $z \in T(y)$ and $p = x \in S(z) = S(m_i)$. Thus, $a(x) t_y(z) s_z(x) = a(p) t_y(m_i) s_{m_i}(p) \in W \Rightarrow Q \subset W$. Conversely, choose an element $a(p) t_y(n) s_n(p)$ from W. We must show that $\exists\ z \in T(y)$ and $x \in S(z)$ such that $z = n$ and $x = p$. Since $a(p) t_y(n) s_n(p) \in W$, we know that $n \in T(y)$ where $p \in S(n)$. Simply set $z = n$, $x = p$ and we are done.

Q.E.D.

Corollary 4.7.38. $(a \circledwedge s) \circledwedge t = a \circledwedge (s \circledwedge t)$, for $a \in R_{\geq 0}^{X}$, $s \in R_{Z \to X}^{>0}$, and $t \in R_{Y \to Z}^{>0}$.

Lemma 4.7.39. $a \circledvee (s \vee t) = (a \circledvee s) \vee (a \circledvee t)$, for $a \in R_{\geq 0}^{X}$, $s, t \in R_{Y \to X}^{>0}$.

Proof: Let $r = s \vee t$, $b = a \circledvee (s \vee t) = a \circledvee r$, and let $c = (a \circledvee s) \vee (a \circledvee t)$. Let $y \in Y$. We must show $b(y) = c(y)$.

We have $b(y) = \bigvee_{x \in R(y)} a(x) r_y(x) = \bigvee_{x \in S(y) \cup T(y)} a(x) [s_y(x) \vee t_y(x)] =$

$$\left\{ \bigvee_{x \in S(y) \setminus T(y)} a(x) [s_y(x) \vee t_y(x)] \right\} \vee \left\{ \bigvee_{z \in T(y) \setminus S(y)} a(z) [s_y(z) \vee t_y(z)] \right\} \vee \left\{ \bigvee_{w \in S(y) \cap T(y)} a(w) [s_y(w) \vee t_y(w)] \right\}$$

$$= \left\{ \bigvee_{x \in S(y)} \{ a(x) s_y(x) \} \right\} \vee \left\{ \bigvee_{z \in T(y)} \{ a(z) t_y(z) \} \right\} \vee \left\{ \bigvee_{w \in S(y) \cap T(y)} \{ a(w) s_y(w) \vee a(w) t_y(w) \} \right\}$$

Note that of these three terms, the last,

$\bigvee_{w \in S(y) \cap T(y)} \{ a(w) s_y(w) \vee a(w) t_y(w) \}$, can be written as

$\left\{\bigvee_{w\in S(y)\cap T(y)}\{a(w)s_y(w)\}\right\}\vee\left\{\bigvee_{w\in S(y)\cap T(y)}\{a(w)t_y(w)\}\right\}$, and since

$\bigvee_{w\in S(y)\cap T(y)}\{a(w)s_y(w)\}\le\bigvee_{w\in S(y)}\{a(w)s_y(w)\}$ and $\bigvee_{w\in S(y)\cap T(y)}\{a(w)t_y(w)\}\le$ $\bigvee_{w\in T(y)}\{a(w)t_y(w)\}$, this third and last term is redundant and can be omitted. Hence

$$b(y)=\left\{\bigvee_{x\in S(y)}\{a(x)s_y(x)\}\right\}\vee\left\{\bigvee_{z\in T(y)}\{a(z)t_y(z)\}\right\}=c(y).$$

Q.E.D.

Corollary 4.7.40. $a\oslash(s\wedge t)=(a\oslash s)\wedge(a\oslash t)$, for $a\in R^{X}_{\ge 0}$, $s,t\in R^{>0}_{Y\to X}$.

Lemma 4.7.41. Let $a\in R^{X}_{\ge 0}$, $t\in R^{>0}_{Y\to X}$. Then $a\oplus t=a\otimes t=a\oslash t\iff t$ is a one point template.

Proof: At $y\in Y$, $a\oplus t$ has gray value $\sum_{x\in T(y)}a(x)t_y(x)$, $a\otimes t$ has gray value $\bigvee_{x\in T(y)}a(x)t_y(x)$, and $a\oslash t$ has gray value $\bigwedge_{x\in T(y)}a(x)t_y(x)$. If t is a 1-point template, then obviously all three above images have the same value, namely $a(x)t_y(x)$. Conversely, if $\sum_{x\in T(y)}a(x)t_y(x)=\bigvee_{x\in T(y)}a(x)t_y(x)=\bigwedge_{x\in T(y)}a(x)t_y(x)$, and if $T(y)$ contains more than one point, we will not necessarily have equality. For example, choosing $t\in R^{>0}_{X\to X}$ to be as in Figure 27,

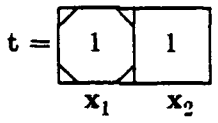


Figure 27. Example for Proof of Lemma 4.7.41

gives us $a(x_1)+a(x_2)$ for the first image, $\vee\{a(x_1),a(x_2)\}$ for the second image, and $\wedge\{a(x_1),a(x_2)\}$ for the third image. All three are obviously not true in the general case for every a. Choosing $a(x_1)=1$, $a(x_2)=2$ will give us $1+2\ne 2\ne 1$.

Q.E.D.

Lemma 4.7.42. $a\ge(a\oslash t^{-1})\otimes t'$, where $a\in R^{X}_{\ge 0}$, $t\in R^{\ge 1}_{X\to X}$, and $y\in T(y)$. The templates t^{-1} and t' are defined in the following way: $t^{-1}=(T,s)$ where

$s_y(x) = \begin{cases} 1/t_x(y) & \text{if } x \in T(y) \\ 0 & \text{if } x \notin T(y) \end{cases}$. In general, the reflection t' is defined as follows: Let $t = (T, t) \in R_{X \to X}$. Then $t' = (T', t')$ is defined to be

$$T'(y) = \{z \in X : z = 2y - x,\ x \in T(y)\}$$

and

$$t'_y(x) = \begin{cases} t_y(<2y-x>) & x \in T(y) \\ 0 & x \notin T(y) \end{cases}.$$

This amounts to flipping the template about its target point. An example is given in Figure 28.

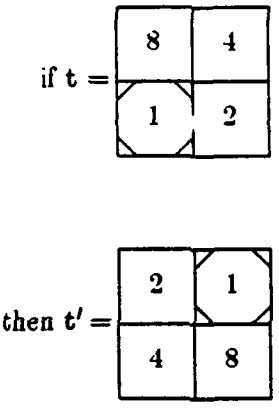


Figure 28. Template t and its Reflection t'

Proof: Let $x \in X$ and write $T(x) = \{y_0, \ldots, y_n\}$ where $x = y_0$. If $w \in X$ and Φ: $Z^k \to Z^k$ is a translation, then when writing $T(w) = \{z_0, \ldots, z_n\}$, use the convention that $\Phi(y_i) = z_i$, that is, the ordering of the z_i's with respect to w is the same as the ordering of the y_i's with respect to x. We shall use the following notation: if $T(y_i) = \{z_0, \ldots, z_n\}$ and $y_i \in T(x)$, write $z_j = y_{ij}$ and $t_{y_i}(z_j) = t_{y_i}(y_{ij})$. Thus, $t_{y_0}(y_{0k}) = t_x(y_k)$, k = 0,...,n, for example. Let $b = a \textcircled{\scriptsize A} t^{-1}$. Then $b(y_i) = \bigwedge_{j=0}^{n} \{a(y_{ij}) \cdot \frac{1}{t_{y_i}(y_{ij})}\}$, i = 0,...,n. Thus, the gray value at location x of $(a \textcircled{\scriptsize A} t^{-1}) \textcircled{\scriptsize V} t'$ is $\bigvee_{i=0}^{n} b(y_i) \cdot t'_{y_0}(y_{0i}) = b(y_k) \cdot t'_{y_0}(y_{0k}) = b(y_k) \cdot t'_x(y_k)$ for some k. Also, we know that

$\mathbf{b}(\mathbf{y}_k) = \bigwedge_{j=0}^{n} \mathbf{a}(\mathbf{y}_{kj}) \cdot \frac{1}{t_{\mathbf{y}_k}(\mathbf{y}_{kj})} = \mathbf{a}(\mathbf{y}_{ki}) \cdot \frac{1}{t_{\mathbf{y}_k}(\mathbf{y}_{ki})}$ for some i. Since $\mathbf{y}_k \in \tau(\mathbf{x})$, we know that $\mathbf{x} \in \tau'(\mathbf{y}_k)$, which means $\exists\ m \in \{0,1,...,n\}$ such that $\mathbf{x} = \mathbf{y}_{km}$. Thus, we know that $t_{\mathbf{y}_k}(\mathbf{y}_{km}) = t_{\mathbf{y}_k}(\mathbf{x}) = t'_{\mathbf{x}}(\mathbf{y}_k)$ by properties of t'. Also, note that $\mathbf{a}(\mathbf{y}_{km}) = \mathbf{a}(\mathbf{x}) = \mathbf{a}(\mathbf{y}_0)$. Since $\mathbf{b}(\mathbf{y}_k) = \mathbf{a}(\mathbf{y}_{ki}) \cdot \frac{1}{t_{\mathbf{y}_k}(\mathbf{y}_{ki})} \leq \mathbf{a}(\mathbf{y}_{km}) \cdot \frac{1}{t_{\mathbf{y}_k}(\mathbf{y}_{km})} = \mathbf{b}(\mathbf{y}_m)$, we have, *for the gray value of* $(\mathbf{a} \oslash \mathbf{t}^{-1}) \otimes \mathbf{t}'$ *at location* $\mathbf{x}$:

$\mathbf{b}(\mathbf{y}_k) \cdot t'_{\mathbf{x}}(\mathbf{y}_k) = \mathbf{a}(\mathbf{y}_{ki}) \cdot \frac{1}{t_{\mathbf{y}_k}(\mathbf{y}_{ki})} \cdot t'_{\mathbf{x}}(\mathbf{y}_k) \leq \mathbf{a}(\mathbf{y}_{km}) \cdot \frac{1}{t_{\mathbf{y}_k}(\mathbf{y}_{km})} \cdot t'_{\mathbf{x}}(\mathbf{y}_k) = \mathbf{a}(\mathbf{x}) \cdot \frac{1}{t_{\mathbf{y}_k}(\mathbf{x})} \cdot t'_{\mathbf{x}}(\mathbf{y}_k)$

$= \mathbf{a}(\mathbf{x}) \cdot \frac{1}{t'_{\mathbf{x}}(\mathbf{y}_k)} \cdot t'_{\mathbf{x}}(\mathbf{y}_k) = \mathbf{a}(\mathbf{x})$, as was desired to show.

Q.E.D.

Corollary 4.7.43. $\mathbf{a} \leq (\mathbf{a} \otimes \mathbf{t}') \oslash \mathbf{t}^{-1}$, where $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$, $\mathbf{t} \in \mathbf{R}^{\geq 1}_{X \to X}$, and $\mathbf{y} \in \tau(\mathbf{y})$.

The proof of this is similar to the proof of Lemma 4.7.42 and can be found in Reference 29.

Lemma 4.7.44. $(\mathbf{a} + \mathbf{b}) \otimes \mathbf{t} \leq \mathbf{a} \otimes \mathbf{t} + \mathbf{b} \otimes \mathbf{t}$, for $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$, and $\mathbf{t} \in \mathbf{R}^{>0}_{Y \to X}$.

Proof: Let $\mathbf{c} = (\mathbf{a} + \mathbf{b}) \otimes \mathbf{t}$, and let $\mathbf{d} = \mathbf{a} \otimes \mathbf{t} + \mathbf{b} \otimes \mathbf{t}$. We must show that for $\mathbf{y} \in \mathbf{Y}$ that $\mathbf{c}(\mathbf{y}) = \mathbf{d}(\mathbf{y})$.

We have $\mathbf{c}(\mathbf{y}) = \bigvee_{\mathbf{x} \in \tau(\mathbf{y})} [\mathbf{a}(\mathbf{x}) + \mathbf{b}(\mathbf{x})] t_{\mathbf{y}}(\mathbf{x}) = \bigvee_{\mathbf{x} \in \tau(\mathbf{y})} [\mathbf{a}(\mathbf{x}) t_{\mathbf{y}}(\mathbf{x}) + \mathbf{b}(\mathbf{x}) t_{\mathbf{y}}(\mathbf{x})]$

$\leq \left\{ \bigvee_{\mathbf{x} \in \tau(\mathbf{y})} [\mathbf{a}(\mathbf{x}) t_{\mathbf{y}}(\mathbf{x})] \right\} + \left\{ \bigvee_{\mathbf{x} \in \tau(\mathbf{y})} [\mathbf{b}(\mathbf{x}) t_{\mathbf{y}}(\mathbf{x})] \right\} = \mathbf{d}(\mathbf{y})$.

Q.E.D.

Lemma 4.7.45. $(\mathbf{a} + \mathbf{b}) \oslash \mathbf{t} \leq \mathbf{a} \oslash \mathbf{t} + \mathbf{b} \oslash \mathbf{t}$, for $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$, and $\mathbf{t} \in \mathbf{R}^{>0}_{Y \to X}$.

Lemma 4.7.46. $\mathbf{a} \oplus (\mathbf{t} \vee \mathbf{s}) + \mathbf{a} \oplus (\mathbf{t} \wedge \mathbf{s}) = \mathbf{a} \oplus (\mathbf{t} + \mathbf{s})$, for $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$, and $\mathbf{s},\mathbf{t} \in \mathbf{R}^{>0}_{Y \to X}$.

Proof: Let $\mathbf{b} = \mathbf{a} \oplus (\mathbf{t} \vee \mathbf{s}) + \mathbf{a} \oplus (\mathbf{t} \wedge \mathbf{s})$ and $\mathbf{c} = \mathbf{a} \oplus (\mathbf{t} + \mathbf{s})$. Then

$$\mathbf{b}(\mathbf{y}) = \sum_{\mathbf{x} \in \tau(\mathbf{y}) \cup \sigma(\mathbf{y})} \mathbf{a}(\mathbf{x}) [\, t_{\mathbf{y}}(\mathbf{x}) \vee s_{\mathbf{y}}(\mathbf{x}) \,] + \sum_{\mathbf{x} \in \tau(\mathbf{y}) \cup \sigma(\mathbf{y})} \mathbf{a}(\mathbf{x}) [\, t_{\mathbf{y}}(\mathbf{x}) \wedge s_{\mathbf{y}}(\mathbf{x}) \,] =$$

$$\sum_{\mathbf{x} \in \tau(\mathbf{y}) \cup \sigma(\mathbf{y})} \mathbf{a}(\mathbf{x}) [\, t_{\mathbf{y}}(\mathbf{x}) \vee s_{\mathbf{y}}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x}) \wedge s_{\mathbf{y}}(\mathbf{x}) \,] = \sum_{\mathbf{x} \in \tau(\mathbf{y}) \cup \sigma(\mathbf{y})} \mathbf{a}(\mathbf{x}) [\, t_{\mathbf{y}}(\mathbf{x}) + s_{\mathbf{y}}(\mathbf{x}) \,] = \mathbf{c}(\mathbf{y}).$$

Here, we use the fact that as a lattice $\mathbf{R}$ has the property $s \vee t + s \wedge t = s + t$, $\forall\ s, t \in \mathbf{R}$.

Q.E.D.

Lemma 4.7.47. $(\mathbf{a}\oplus\mathbf{s})\vee(\mathbf{a}\oplus\mathbf{t}) \leq \mathbf{a}\oplus(\mathbf{s}\vee\mathbf{t})$, for $\mathbf{t} \in \mathbf{R}_{\mathbf{Y}\to\mathbf{X}}^{\geq 0}$ and $\mathbf{a} \in \mathbf{R}_{\geq 0}^{\mathbf{X}}$.

Proof: At location $\mathbf{y} \in \mathbf{Y}$, $(\mathbf{a}\oplus\mathbf{s})\vee(\mathbf{a}\oplus\mathbf{t})$ has gray value

$$\left\{\sum_{\mathbf{x}\in S(\mathbf{y})} \mathbf{a}(\mathbf{x})\mathbf{s}_{\mathbf{y}}(\mathbf{x})\right\} \vee \left\{\sum_{\mathbf{z}\in T(\mathbf{y})} \mathbf{a}(\mathbf{z})\mathbf{t}_{\mathbf{y}}(\mathbf{z})\right\}$$

Note that we can write this quantity as

$$\left\{\sum_{\mathbf{w}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{w})\mathbf{s}_{\mathbf{y}}(\mathbf{w})\right\} \vee \left\{\sum_{\mathbf{z}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{z})\mathbf{t}_{\mathbf{y}}(\mathbf{z})\right\}$$

as if $\mathbf{z} \notin S(\mathbf{y})$, then $\mathbf{s}_{\mathbf{y}}(\mathbf{z}) = 0$, and we have $\mathbf{a}(\mathbf{x})\mathbf{s}_{\mathbf{y}}(\mathbf{x}) = 0$. A similar case holds for any $\mathbf{z} \notin T(\mathbf{y})$. Also, on the right hand side of the equation, the gray value is

$$\sum_{\mathbf{u}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{u})\left[\mathbf{s}_{\mathbf{y}}(\mathbf{u})\vee\mathbf{t}_{\mathbf{y}}(\mathbf{u})\right].$$

Note that

$$\mathbf{a}(\mathbf{u})\mathbf{s}_{\mathbf{y}}(\mathbf{u}) \leq \mathbf{a}(\mathbf{u})\left[\mathbf{s}_{\mathbf{y}}(\mathbf{u})\vee\mathbf{t}_{\mathbf{y}}(\mathbf{u})\right]$$
$$\mathbf{a}(\mathbf{u})\mathbf{t}_{\mathbf{y}}(\mathbf{u}) \leq \mathbf{a}(\mathbf{u})\left[\mathbf{s}_{\mathbf{y}}(\mathbf{u})\vee\mathbf{t}_{\mathbf{y}}(\mathbf{u})\right]$$

as $\mathbf{a}(\mathbf{u})$ and $\mathbf{t}_{\mathbf{y}}(\mathbf{u}) \geq 0$. This implies that

$$\sum_{\mathbf{u}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{u})\mathbf{s}_{\mathbf{y}}(\mathbf{u}) \leq \sum_{\mathbf{u}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{u})\left[\mathbf{s}_{\mathbf{y}}(\mathbf{u})\vee\mathbf{t}_{\mathbf{y}}(\mathbf{u})\right]$$

and

$$\sum_{\mathbf{u}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{u})\mathbf{t}_{\mathbf{y}}(\mathbf{u}) \leq \sum_{\mathbf{u}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{u})\left[\mathbf{s}_{\mathbf{y}}(\mathbf{u})\vee\mathbf{t}_{\mathbf{y}}(\mathbf{u})\right]$$

Thus,

$$\left\{\sum_{\mathbf{w}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{w})\mathbf{s}_{\mathbf{y}}(\mathbf{w})\right\} \vee \left\{\sum_{\mathbf{z}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{z})\mathbf{t}_{\mathbf{y}}(\mathbf{z})\right\} \leq \sum_{\mathbf{u}\in S(\mathbf{y})\cup T(\mathbf{y})} \mathbf{a}(\mathbf{u})\left[\mathbf{s}_{\mathbf{y}}(\mathbf{u})\vee\mathbf{t}_{\mathbf{y}}(\mathbf{u})\right]$$

Q.E.D.

Lemma 4.7.48. $\mathbf{s} \leq \mathbf{t} \Rightarrow \mathbf{a}\oplus\mathbf{s} \leq \mathbf{a}\oplus\mathbf{t}$, for $\mathbf{s}, \mathbf{t} \in \mathbf{R}_{\mathbf{Y}\to\mathbf{X}}^{\geq 0}$ and $\mathbf{a} \in \mathbf{R}_{\geq 0}^{\mathbf{X}}$.

Proof: At location $\mathbf{y} \in \mathbf{Y}$, $\mathbf{a}\oplus\mathbf{s}$ has gray value $\sum_{\mathbf{x}\in S(\mathbf{y})} \mathbf{a}(\mathbf{x})\mathbf{s}_{\mathbf{y}}(\mathbf{x})$. Since $\mathbf{s} \leq \mathbf{t}$, we know $S(\mathbf{y}) \subset T(\mathbf{y})$, which implies that

$\sum_{\mathbf{x}\in S(\mathbf{y})} \mathbf{a}(\mathbf{x})\mathbf{s}_{\mathbf{y}}(\mathbf{x}) = \sum_{\mathbf{x}\in S(\mathbf{y})} \mathbf{a}(\mathbf{x})\mathbf{s}_{\mathbf{y}}(\mathbf{x}) + \sum_{\mathbf{z}\in T(\mathbf{y})\setminus S(\mathbf{y})} \mathbf{a}(\mathbf{z})\mathbf{s}_{\mathbf{y}}(\mathbf{z})$, which is true as $\mathbf{a}(\mathbf{z})\mathbf{s}_{\mathbf{y}}(\mathbf{z}) = 0$

$\forall \in z \in \mathcal{T}(y) \backslash \mathcal{S}(y)$. Thus, since $s_y \leq t_y \; \forall \, y \in Y$,

$\sum_{x \in \mathcal{S}(y)} a(x) s_y(x) + \sum_{z \in \mathcal{T}(y) \backslash \mathcal{S}(y)} a(z) s_y(z) \leq \sum_{x \in \mathcal{S}(y)} a(x) t_y(x) + \sum_{z \in \mathcal{T}(y) \backslash \mathcal{S}(y)} a(z) t_y(z) =$

$\sum_{x \in \mathcal{S}(y) \bigcup \mathcal{T}(y)} a(x) t_y(x) = \sum_{x \in \mathcal{T}(y)} a(x) t_y(x)$, which is the gray value of $\mathbf{a} \oplus \mathbf{t}$ at location $y \in Y$.

Q.E.D.

Corollary 4.7.49. $\mathbf{s} \leq \mathbf{t} \Rightarrow \mathbf{a} \circledcirc \mathbf{s} \leq \mathbf{a} \circledcirc \mathbf{t}$, for $\mathbf{s}, \mathbf{t} \in \mathbf{R}^{\geq 0}_{Y \to X}$ and $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$.

Corollary 4.7.50. $\mathbf{s} \leq \mathbf{t} \Rightarrow \mathbf{a} \circledast \mathbf{s} \leq \mathbf{a} \circledast \mathbf{t}$, for $\mathbf{s}, \mathbf{t} \in \mathbf{R}^{\geq 0}_{Y \to X}$ and $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$.

The proofs of these last two corollaries are similar to that of Lemma 4.7.48, and can be found in Reference 29.

c. Applications and Examples of $\circledcirc$ Properties

A few applications of the preceding results are now presented. Suppose a template $\mathbf{t}$ can be decomposed into templates $\mathbf{t}_i$ such that

$$\mathbf{t} = \mathbf{t}_1 \circledcirc \mathbf{t}_2 \circledcirc \cdots \circledcirc \mathbf{t}_n.$$

Then $\mathbf{a} \circledcirc \mathbf{t} = \mathbf{a} \circledcirc (\mathbf{t}_1 \circledcirc \mathbf{t}_2 \circledcirc \cdots \circledcirc \mathbf{t}_n) =$

$$((...((\mathbf{a} \circledcirc \mathbf{t}_1) \circledcirc \mathbf{t}_2 \circledcirc)...) \circledcirc \mathbf{t}_n)$$

Computing $\mathbf{a} \circledcirc \mathbf{t}$ using the above equation results, in most cases, in fewer multiplications than computing $\mathbf{a} \circledcirc \mathbf{t}$ directly.

Now suppose one cannot find templates $\mathbf{t}_i$ such that

$$\mathbf{t} = \mathbf{t}_1 \circledcirc \mathbf{t}_2 \circledcirc \cdots \circledcirc \mathbf{t}_n,$$

but it is known that $\mathbf{t} = \mathbf{r} \vee \mathbf{s}$, where $\mathbf{r}$ and $\mathbf{s}$ can be decomposed as follows:

$$\mathbf{r} = \mathbf{r}_1 \circledcirc \mathbf{r}_2 \circledcirc \cdots \circledcirc \mathbf{r}_k$$

$$\mathbf{s} = \mathbf{s}_1 \circledcirc \mathbf{s}_2 \circledcirc \cdots \circledcirc \mathbf{s}_m$$

Then

$$\mathbf{a} \circledcirc \mathbf{t} = \mathbf{a} \circledcirc (\mathbf{r} \vee \mathbf{s}) = (\mathbf{a} \circledcirc \mathbf{r}) \vee (\mathbf{a} \circledcirc \mathbf{s})$$

$$= (\mathbf{a} \oslash (\mathbf{r}_1 \oslash \mathbf{r}_2 \oslash \cdots \oslash \mathbf{r}_k)) \bigvee (\mathbf{a} \oslash (\mathbf{s}_1 \oslash \mathbf{s}_2 \oslash \cdots \oslash \mathbf{s}_m))$$

$$= ((...((\mathbf{a} \oslash \mathbf{r}_1) \oslash \mathbf{r}_2 \oslash)...) \oslash \mathbf{r}_k) \bigvee ((...((\mathbf{a} \oslash \mathbf{s}_1) \oslash \mathbf{s}_2 \oslash)...) \oslash \mathbf{s}_m)$$

which is also more efficiently computed in many cases than with $\mathbf{t}$ directly.

Here is an example of this. Let $\mathbf{t} \in \mathbf{R}^{>0}_{X \to X}$ be the template as presented in Figure 29 below.

$\mathbf{t} =$

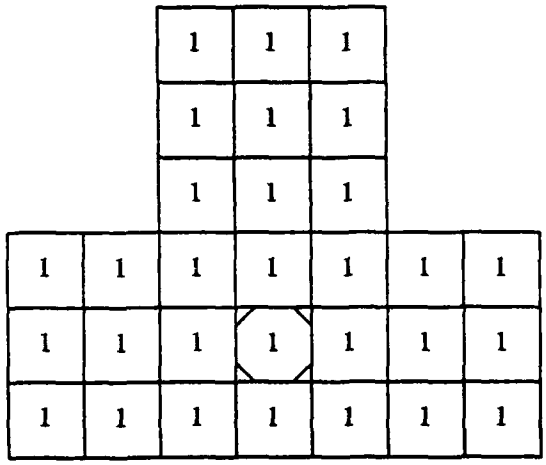
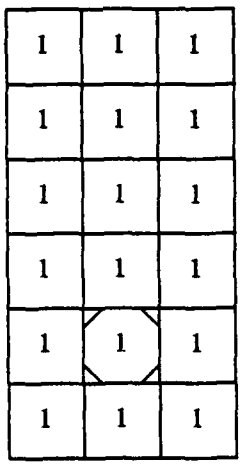


Figure 29. Template $\mathbf{t}$

Then $\mathbf{t}$ can be written as $\mathbf{t} = \mathbf{r} \bigvee \mathbf{s}$ where $\mathbf{r}$ and $\mathbf{s}$ are as in Figure 30.

$\mathbf{r} =$



$\mathbf{s} =$

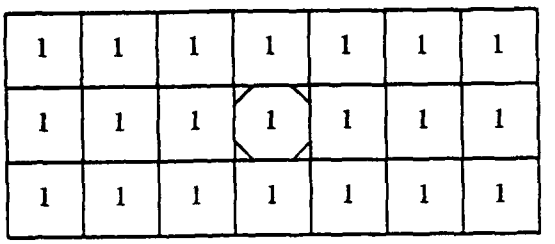
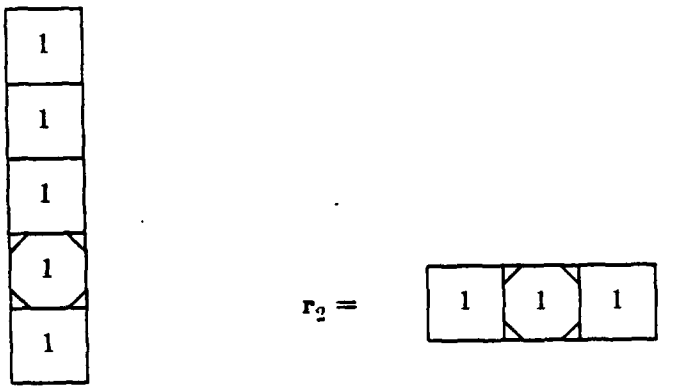


Figure 30. Templates $\mathbf{r}$ and $\mathbf{s}$

Now $\mathbf{r}$ and $\mathbf{s}$ can be written as $\mathbf{r} = \mathbf{r}_1 \circledcirc \mathbf{r}_2$, $\mathbf{s} = \mathbf{s}_1 \circledcirc \mathbf{s}_2$, where $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{s}_1$, and $\mathbf{s}_2$ are as in Figure 31.

$\mathbf{r}_1 =$ $\mathbf{r}_2 =$



$\mathbf{s}_1 =$ $\mathbf{s}_2 =$

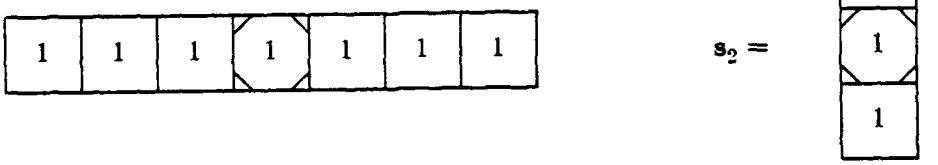


Figure 31. Templates $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{s}_1$, and $\mathbf{s}_2$

Thus, given a binary image in which we wished to locate an object which at least is the size of our template, we could replace the algorithm $\mathbf{a} \oslash \mathbf{t}$ by the last expression in the equation

$$\mathbf{a} \oslash \mathbf{t} = \mathbf{a} \oslash (\mathbf{r} \vee \mathbf{s}) = (\mathbf{a} \oslash \mathbf{r}) \vee (\mathbf{a} \oslash \mathbf{s}) =$$

$$((\mathbf{a} \oslash \mathbf{r}_1) \oslash \mathbf{r}_2) \vee ((\mathbf{a} \oslash \mathbf{s}_1) \oslash \mathbf{s}_2)$$

The number of multiplications per pixel using the template $\mathbf{t}$ is 30, while the number of multiplications per pixel using the templates $\mathbf{r}$ and $\mathbf{s}$ is 18.

The Division Algorithm. A Euclidean domain is defined to be an integral domain D with a Euclidean valuation on D. A Euclidean valuation on an integral domain is a function v: $D\backslash\{0\} \longrightarrow \{0, 1, 2, \cdots\}$ such that the following conditions are satisfied:

(1) For all a, b $\in$ D with b $\neq 0$, there exist q and r in D such that $a = qb + r$, where either $r = 0$ or $v(r) < v(b)$.

(2) For all a, b $\in$ D, where neither a nor b is 0, $v(a) \leq v(ab)$.

For example, the integers $\mathbf{Z}$ are a Euclidean domain, as given a, t $\in \mathbf{Z}$, there exist unique integers q and r such that

$$a = qt + r \text{ , where } r < t.$$

As another example, take the polynomials in 1 variable. Given polynomials a(x), t(x) with coefficients in a field F, there exist unique polynomials q(x) and r(x) such that

$$a(x) = q(x)t(x) + r(x) \text{ , where degree}(r(x)) < \text{degree}(t(x)).$$

While the structure of templates and images under $\oslash$ and $\vee$ comes nowhere close to a euclidean domain, it has been possible to construct a division algorithm within the structures $\oslash$ and $\boxtimes$.

For our purposes we will use a valuation function defined on a lattice L, Reference 28.

Definition. A valuation on a lattice L is a real valued function (functional) $v[x]$ on L which satisfies

$$v(\mathbf{a}) + v(\mathbf{b}) = v(\mathbf{a} \wedge \mathbf{b}) + v(\mathbf{a} \vee \mathbf{b}).$$

Given an image $\mathbf{a}$ and a template $\mathbf{t}$ we can define a division algorithm for $\mathbf{a}$ with basis $\mathbf{t}$ with respect to $\oslash$ and $\vee$. We define a valuation function on the lattice $(\mathbf{R}^{\times}_{\geq 0}, \vee)$ by

$$v(\mathbf{a}) = \vee\mathbf{a} - \wedge\mathbf{a}.$$

Let E(P) denote the sublattice of $E(\mathbf{R}^{\geq 0}_{X \to X})$ which includes only invariant templates, that is, $E(P) = \{ \mathbf{t} \in E(\mathbf{R}^{\geq 0}_{X \to X}) : \mathbf{t} \text{ is invariant} \}$. (Note that we are abusing notation here, and using $\mathbf{t} \in [\mathbf{t}]$ to represent $[\mathbf{t}]$, an equivalence class.) We can define an induced valuation function on E(P) by

$$v\colon E(P) \to \mathbf{R}\ ,\quad v(\mathbf{t}) \equiv v(\tilde{t}_x)$$

where $\tilde{t}$ is the second coordinate of $\tilde{\mathbf{t}} = (\tilde{T}, \tilde{t})$, the unique extension of $\mathbf{t}$ to $\mathbf{R}^n$. Since $\tilde{t}_x$ is an image, this is well defined.

We present a few examples of this in the following figure.

$$\mathbf{a} = \begin{matrix} 5 & 3 & 9 & 6 & 9 & 9 & 7 \\ 3 & 3 & 9 & 7 & 3 & 9 & 8 \\ 2 & 9 & 1 & 6 & 4 & 9 & 4 \\ 2 & 2 & 9 & 7 & 9 & 6 & 5 \\ 5 & 2 & 9 & 5 & 8 & 9 & 7 \\ 3 & 1 & 5 & 6 & 8 & 4 & 9 \end{matrix}$$

$$v(\mathbf{a}) = 8$$

$$t = \begin{matrix} & 1 & \\ 1 & 19 & 1 \\ & 1 & \end{matrix}$$

$$t_x = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 19 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$$v(t_x) = 19$$

Figure 32. Examples of Valuation Function

The Division Algorithm for $\textcircled{\vee}$. Given an image $\mathbf{a} \in \mathbf{R}^{X}_{\geq 0}$ and an invariant template $\mathbf{t} \in \mathbf{R}^{>0}_{X-X}$ where $\mathbf{x} \in T(\mathbf{x})$, then there exist images $\mathbf{q}$ and $\mathbf{r}$ such that

$$\mathbf{a} = (\mathbf{q} \textcircled{\vee} \mathbf{t}') \vee \mathbf{r}$$

In other words, an expression similar to the division algorithm ($a = q*t + r$) holds for $\mathbf{a}$ and $\mathbf{t}$. Here, we have $v((\mathbf{r}_k \textcircled{\wedge} \mathbf{t}^{-1}) \textcircled{\vee} \mathbf{t}') \leq v(\mathbf{t})$, where v is the valuation function above. Thus, in this sense, the remainder $\mathbf{r}$ is small with respect to a valuation function.

It has been shown, Reference 29, that choosing $\mathbf{q} = \mathbf{a} \textcircled{\wedge} \mathbf{t}^{-1}$, and $\mathbf{r} = \mathbf{a} * [\chi_{>0}(\mathbf{a} - (\mathbf{a} \textcircled{\wedge} \mathbf{t}^{-1}) \textcircled{\vee} \mathbf{t}')]$ will make the above equality true.

One useful result of being able to express $\mathbf{a}$ in this way is in the following application. Making the following recursive definitions, we can write $\mathbf{a}$ in a series form

$$\mathbf{a} = \mathbf{r}_0 \vee (\mathbf{r}_1 \textcircled{\vee} \mathbf{t}') \vee (\mathbf{r}_2 \textcircled{\vee} (\mathbf{t}')^2) \vee \cdots \vee (\mathbf{r}_{n-1} \textcircled{\vee} (\mathbf{t}')^{n-1}) \vee (\mathbf{a}_n \textcircled{\vee} (\mathbf{t}')^n)$$

$$= \left\{ \bigvee_{k=0}^{n-1} \mathbf{r}_k \textcircled{\vee} (\mathbf{t}')^k \right\} \vee (\mathbf{a}_n \textcircled{\vee} (\mathbf{t}')^n)$$

where

$\mathbf{a}_0 \equiv \mathbf{a}$,

$\mathbf{r}_k = \mathbf{a}_k * [\chi_{>0}(\mathbf{a}_k - (\mathbf{a}_k \textcircled{\wedge} \mathbf{t}^{-1}) \textcircled{\vee} \mathbf{t}')], k = 0,\ldots,n-1$,

$\mathbf{a}_k = \mathbf{a}_{k-1} \textcircled{\wedge} \mathbf{t}^{-1}, k = 1,\ldots,n$,

and

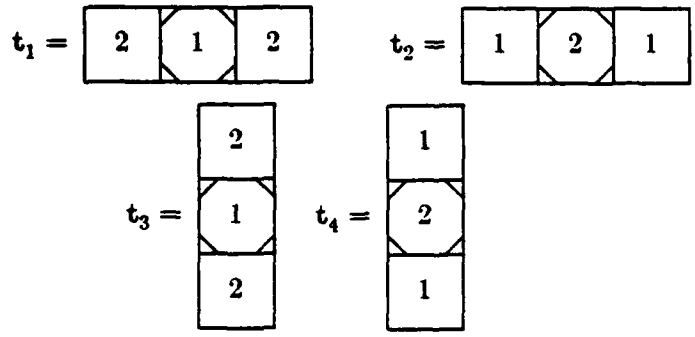
$(\mathbf{t}')^k \equiv \mathbf{t}' \textcircled{\vee} \mathbf{t}' \cdots \textcircled{\vee} \mathbf{t}'$, k times.

An Image Blurring Technique. During our investigation concerning the properties $\textcircled{\vee}$, several novel blurring techniques which emphasize hot targets and blurs streaking in infrared images were discovered. Here we present one of these blurring transforms.

The Image Algebra equation is

$$[(\mathbf{a} \textcircled{\vee} \mathbf{t}_1 - \mathbf{a} \textcircled{\wedge} \mathbf{t}_2)^2 + (\mathbf{a} \textcircled{\vee} \mathbf{t}_3 - \mathbf{a} \textcircled{\wedge} \mathbf{t}_4)^2]^{1/2}$$

with the templates $\mathbf{t}_i$'s as below.



A pictorial example is given in the next two figures.

Figure 33. Input Image **a**

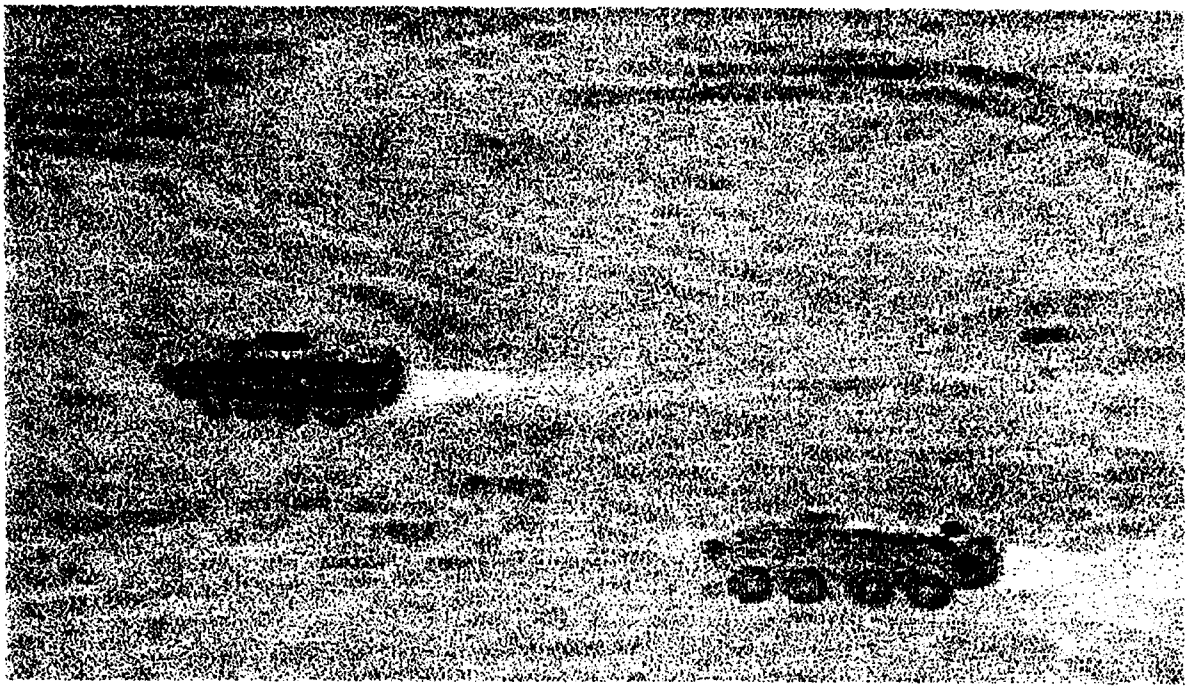


Figure 34. Blurred (Transformed) Image

8. THE ALGEBRA DETERMINED BY $\boxtimes$

The algebraic structure determined by $\boxtimes$ is mathematically not as rich as the one under $\circledcirc$, but it does yield some surprising results, such as a division algorithm similar to the one for $\circledcirc$.

a. Operations Between Templates under $\boxtimes$

As was for the case of the algebra determined by $\circledcirc$, the results are presented for the most general situation possible, that is, for templates from $\mathbf{Y}$ to $\mathbf{X}$.

Two very basic results are Lemmas 4.8.1 and 4.8.2:

Lemma 4.8.1. $\mathbf{s} \boxtimes \mathbf{t} = -(-\mathbf{s} \boxtimes -\mathbf{t})$ for $\mathbf{s} \in \mathbf{R}^{>0}_{\mathbf{Z}\to\mathbf{X}}$, $\mathbf{t} \in \mathbf{R}^{>0}_{\mathbf{Y}\to\mathbf{Z}}$.

Proof: Both templates are members of $\mathbf{R}^{>0}_{\mathbf{Y}\to\mathbf{X}}$, and have the same configuration. We note that since the template weights are added and not multiplied, as in the analogous case for $\circledcirc$, that we introduce a minus sign on both of the templates on the right hand side. On the left hand side of the equality the image at location $\mathbf{y} \in \mathbf{Y}$ has form

$$\{(\mathbf{x}, r_{\mathbf{y}}(\mathbf{x})) : r_{\mathbf{y}}(\mathbf{x}) = \bigwedge_{\mathbf{w} \in T(\mathbf{y})} \{ t_{\mathbf{y}}(\mathbf{w}) + s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \} \}.$$

On the right hand side of the equality the image at location $\mathbf{y} \in \mathbf{Y}$ has form

$$\{(\mathbf{x}, u_{\mathbf{y}}(\mathbf{x})) : u_{\mathbf{y}}(\mathbf{x}) = -(\bigvee_{\mathbf{w} \in T(\mathbf{y})} \{ -t_{\mathbf{y}}(\mathbf{w}) + -s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \}) \}.$$

Similar to the reasoning in the proof of Lemma 4.7.1 we can conclude that

$$r_{\mathbf{y}}(\mathbf{x}) = \bigwedge_{\mathbf{w} \in T(\mathbf{y})} \{ -[-t_{\mathbf{y}}(\mathbf{w}) + -s_{\mathbf{w}}(\mathbf{x})] : \mathbf{x} \in S(\mathbf{w}) \} =$$

$$- \bigvee_{\mathbf{w} \in T(\mathbf{y})} \{ -t_{\mathbf{y}}(\mathbf{w}) + -s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \} = u_{\mathbf{y}}(\mathbf{x}).$$

Q.E.D.

Lemma 4.8.2. $\mathbf{s} \boxtimes \mathbf{t} \leq \mathbf{s} \boxtimes \mathbf{t}$ for $\mathbf{s} \in \mathbf{R}^{>0}_{\mathbf{Z} \to \mathbf{X}}$, $\mathbf{t} \in \mathbf{R}^{>0}_{\mathbf{Y} \to \mathbf{Z}}$.

Proof: We have to show that for $\mathbf{r} = \mathbf{s} \boxtimes \mathbf{t}$ and for $\mathbf{u} = \mathbf{s} \boxtimes \mathbf{t}$ that $r_{\mathbf{y}}(\mathbf{x}) \leq u_{\mathbf{y}}(\mathbf{x})$. We have

$$r_{\mathbf{y}}(\mathbf{x}) = \bigwedge_{\mathbf{w} \in T(\mathbf{y})} \{ t_{\mathbf{y}}(\mathbf{w}) + s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \}$$

$$\leq \bigvee_{\mathbf{w} \in T(\mathbf{y})} \{ t_{\mathbf{y}}(\mathbf{w}) + s_{\mathbf{w}}(\mathbf{x}) : \mathbf{x} \in S(\mathbf{w}) \} = u_{\mathbf{y}}(\mathbf{x}).$$

Q.E.D.

We also have the associativity of templates under $\boxtimes$:

Theorem 4.8.3. $(\mathbf{s} \boxtimes \mathbf{t}) \boxtimes \mathbf{r} = \mathbf{s} \boxtimes (\mathbf{t} \boxtimes \mathbf{r})$ for $\mathbf{r} \in \mathbf{R}^{>0}_{\mathbf{W} \to \mathbf{Z}}$, $\mathbf{t} \in \mathbf{R}^{>0}_{\mathbf{Z} \to \mathbf{Y}}$, $\mathbf{s} \in \mathbf{R}^{>0}_{\mathbf{Y} \to \mathbf{X}}$.

Proof: The proof of this is very similar to that of Theorem 4.7.4, and we omit it from this report. However, it is included in Reference 29.

Corollary 4.8.4. $(\mathbf{s} \boxtimes \mathbf{t}) \boxtimes \mathbf{r} = \mathbf{s} \boxtimes (\mathbf{t} \boxtimes \mathbf{r})$ for $\mathbf{r} \in \mathbf{R}^{>0}_{\mathbf{W} \to \mathbf{Z}}$, $\mathbf{t} \in \mathbf{R}^{>0}_{\mathbf{Z} \to \mathbf{Y}}$, $\mathbf{s} \in \mathbf{R}^{>0}_{\mathbf{Y} \to \mathbf{X}}$.

Thus we know

Corollary 4.8.5. $\{ \mathbf{R}^{>0}_{\mathbf{X} \to \mathbf{X}}, \boxtimes \}$ is a semigroup.

Corollary 4.8.6. $\{ \mathbf{R}^{>0}_{\mathbf{X} \to \mathbf{X}}, \boxtimes \}$ is a semigroup.

Similar to the structure under $\boxtimes$, we have the distributivity of $\boxtimes$ over $\vee$, and their duals:

Lemma 4.8.7. $\mathbf{s} \boxtimes (\mathbf{t} \vee \mathbf{r}) = (\mathbf{s} \boxtimes \mathbf{t}) \vee (\mathbf{s} \boxtimes \mathbf{r})$ if $\mathbf{t}, \mathbf{r} \in \mathbf{R}^{>0}_{\mathbf{Y} \to \mathbf{Z}}$, and $\mathbf{s} \in \mathbf{R}^{>0}_{\mathbf{Z} \to \mathbf{X}}$.

Proof: Let $\mathbf{u} = \mathbf{t} \vee \mathbf{r}$, $\mathbf{k} = \mathbf{s} \boxtimes (\mathbf{t} \vee \mathbf{r}) = \mathbf{s} \boxtimes \mathbf{u}$, $\mathbf{p} = \mathbf{s} \boxtimes \mathbf{t}$, $\mathbf{q} = \mathbf{s} \boxtimes \mathbf{r}$, and $\mathbf{l} = (\mathbf{s} \boxtimes \mathbf{t}) \vee (\mathbf{s} \boxtimes \mathbf{r}) = \mathbf{p} \vee \mathbf{q}$. The proof that the configurations are the same is identical to that found in Theorem 4.7.13. As for the gray values $k_{\mathbf{y}}(\mathbf{x})$ and $l_{\mathbf{y}}(\mathbf{x})$,

we have:

$$k_y(x) = \bigvee \{ u_y(z) + s_z(x) : z \in U(y) \text{ and } x \in S(z) \}$$

$$= \bigvee \{ [t_y(z) \vee r_y(z)] + s_z(x) : z \in T(y) \cup R(y) \text{ and } x \in S(z) \}$$

$$= \bigvee \{ (t_y(z) + s_z(x)) \vee (r_y(z) + s_z(x)) : z \in T(y) \cup R(y) \text{ and } x \in S(z) \},$$

with this last equality following from the fact that the template weights are positive. Continuing,

$$k_y(x) = \bigvee \{ t_y(z) + s_z(x) : z \in T(y) \text{ and } x \in S(z) \} \vee \bigvee \{ r_y(z) + s_z(x) : z \in R(y) \text{ and } x \in S(z) \},$$

as r, s, and t are positive and have zero weight value outside of their respective configurations. This is exactly $l_y(x)$.

Q.E.D.

Corollary 4.8.8. $(t \vee r) \boxtimes s = (t \boxtimes s) \vee (r \boxtimes s)$ if $t, r \in R^{>0}_{Z \to X}$, and $s \in R^{>0}_{Y \to Z}$.

Corollary 4.8.9. $s \boxtimes (t \wedge r) = (s \boxtimes t) \wedge (s \boxtimes r)$ if $t, r \in R^{>0}_{Y \to Z}$, and $s \in R^{>0}_{Z \to X}$.

Corollary 4.8.10. $(t \wedge r) \boxtimes s = (t \boxtimes s) \wedge (r \boxtimes s)$ if $t, r \in R^{>0}_{Z \to X}$, and $s \in R^{>0}_{Y \to Z}$.

By Lemma 4.7.3 and Corollaries 4.8.7 and 4.8.8, we have

Theorem 4.8.11. $\{ R^{>0}_{X \to X}, \leq \vee, \boxtimes \}$ is an m-semilattice.

Corollary 4.8.12. $\{ R^{>0}_{X \to X}, \leq \wedge, \boxtimes \}$ is an m-semilattice.

Since any m-semilattice is an m-poset, we have

Theorem 4.8.13. $\{ R^{>0}_{X \to X}, \leq \vee, \boxtimes \}$ is an m-poset.

Corollary 4.8.14. $\{ R^{>0}_{X \to X}, \leq \wedge, \boxtimes \}$ is an m-poset.

Thus we know that the following two lemmas are true, being properties of an m-poset.

Lemma 4.8.15. $u \leq t \Rightarrow s \boxtimes u \leq s \boxtimes t$ and $u \boxtimes s \leq t \boxtimes s$ for $s, t \in E(R^{>0}_{Z \to Y})$, and $r \in E(R^{>0}_{X \to X})$.

Lemma 4.8.16. $u \leq t \Rightarrow s \boxtimes u \leq s \boxtimes t$ and $u \boxtimes s \leq t \boxtimes s$ for $s, t \in E(R^{>0}_{Z \to Y})$, and $r \in E(R^{>0}_{X \to X})$.

Lemma 4.8.17 $u \leq t \Rightarrow s \boxtimes u \boxtimes r \leq s \boxtimes t \boxtimes r$ for $r \in R^{>0}_{W \to Z}, t \in R^{>0}_{Z \to Y}, s \in R^{>0}_{Y \to X}$.

By Corollary 4.8.5, we have

Theorem 4.8.18. $\{ \mathbf{R}^{\geq 0}_{X \to X}, \leq, \vee, \boxtimes \}$ is a po-semigroup (partially ordered semigroup).

Corollary 4.8.19. $\{ \mathbf{R}^{\geq 0}_{X \to X}, \leq, \wedge, \boxtimes \}$ is a po-semigroup.

It is not obvious how to put a nice equivalence relation on $\mathbf{R}^{\geq 0}_{Y \to X}$ to make it a lattice. In the case of $\circledvee$, the induced operation $\circledvee$ between equivalence classes, $[\mathbf{s}] \circledvee [\mathbf{t}] \equiv [\mathbf{s} \circledvee \mathbf{t}]$, was well-defined. However, using this same equivalence relation, we can find examples of $\mathbf{s}$ and $\mathbf{t}$ where $[\mathbf{s}] \boxtimes [\mathbf{t}] \neq [\mathbf{s} \boxtimes \mathbf{t}]$. Given the templates $\mathbf{s}$, $\mathbf{u}$, and $\mathbf{t}$ as in Figure 35, and the equivalence relation $\sim$ as defined in Section 7 we see that $\mathbf{s} \boxtimes \mathbf{t} = \mathbf{s} \boxtimes \mathbf{u}$ is not necessarily true for all templates, where $\mathbf{t}$ and $\mathbf{u}$ are in the same equivalence class. In Figure 35, we present an example of this.

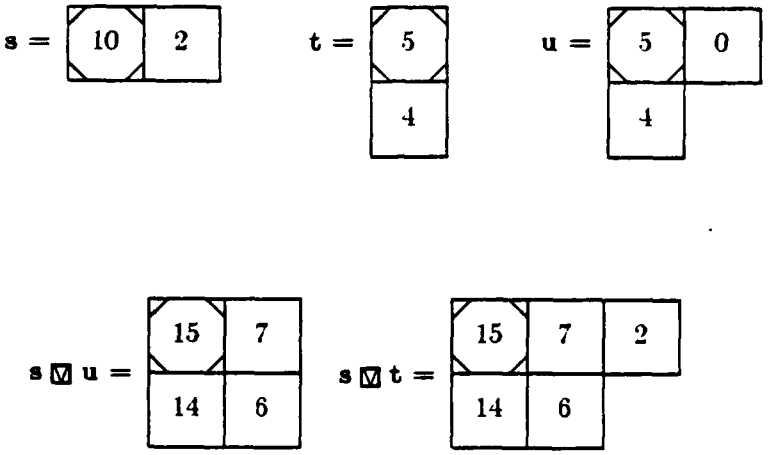


Figure 35. Example of $[\mathbf{s}] \boxtimes [\mathbf{t}] \neq [\mathbf{s} \boxtimes \mathbf{u}]$, $\mathbf{t} \sim \mathbf{u}$

Here we see that for $\mathbf{t}$ and $\mathbf{u}$ in the same equivalence class, $\mathbf{s} \boxtimes \mathbf{t} \neq \mathbf{s} \boxtimes \mathbf{u}$. Thus, using the equivalence relation $\sim$ and the naturally induced operation $\boxtimes$ between equivalence classes, we find that this induced operation is not well defined. There may be another equivalence relation that may induce a well-defined $\boxtimes$ operation between equivalence classes, but it is not as readily apparent. Not having the strength of a lattice structure, but only of a semi-lattice, the structure of the Image Algebra under $\boxtimes$ is not as rich. We state

a few more theorems which may prove to be useful at some future date. Any proofs not given can be found in Reference 29.

Theorem 4.8.20. $\mathbf{s} \boxtimes \mathbf{t} = (\mathbf{t} \boxtimes \mathbf{s})$ for $\mathbf{s},\mathbf{t} \in \mathbf{R}^{>0}_{X\to X}$, and $\mathbf{s}$, $\mathbf{t}$ circulant.

Corollary 4.8.21. $\mathbf{s} \boxtimes \mathbf{t} = (\mathbf{t} \boxtimes \mathbf{s})$ for $\mathbf{s},\mathbf{t} \in \mathbf{R}^{>0}_{X\to X}$, and $\mathbf{s}$, $\mathbf{t}$ circulant.

Theorem 4.8.22. $\{ \mathbf{R}^{>0}_{X\to X}, \leq \vee, \boxtimes, \mathbf{i} \}$ is a po-monoid, where $\mathcal{I}(\mathbf{x}) = \{\mathbf{x}\}$and $i_{\mathbf{x}}(\mathbf{x}) = 1$.

Corollary 4.8.23 $\{ \mathbf{R}^{>0}_{X\to X}, \leq \wedge, \boxtimes, \mathbf{i} \}$ is a po-monoid, where $\mathbf{i}$ is as in Theorem 4.8.22.

b. Image-Template Operations Under $\boxtimes$

As was the case for image-template operations under $\circledcirc$, there is no well-known structure for the image-template operations under $\boxtimes$. Some basic properties that have been found to hold are listed here.

Lemma 4.8.24. $\mathbf{a} \boxtimes \mathbf{s} = -(-\mathbf{a} \boxtimes -\mathbf{s})$, for $\mathbf{a} \in \mathbf{R}^{X}$, $\mathbf{s} \in \mathbf{R}_{Y\to X}$.

Proof: $\mathbf{a} \boxtimes \mathbf{s} = \{(\mathbf{z}, \mathbf{c}(\mathbf{z})) : \mathbf{c}(\mathbf{z}) = \bigwedge\limits_{\mathbf{y}\in S(\mathbf{z})} \mathbf{a}(\mathbf{y}) + s_{\mathbf{z}}(\mathbf{y}),\ \mathbf{y} \in \mathbf{Y}\}$, and $-(-\mathbf{a} \boxtimes -\mathbf{s}) =$

$\{(\mathbf{z}, \mathbf{d}(\mathbf{z}) : \mathbf{d}(\mathbf{z}) = \bigvee\limits_{\mathbf{y}\in S(\mathbf{z})} -\mathbf{a}(\mathbf{y}) + -s_{\mathbf{z}}(\mathbf{y}),\ \mathbf{y} \in \mathbf{Y}\}$. We have

$\mathbf{d}(\mathbf{z}) = \bigvee\limits_{\mathbf{y}\in S(\mathbf{z})} -[\mathbf{a}(\mathbf{y}) + s_{\mathbf{z}}(\mathbf{y})] = -\bigwedge\limits_{\mathbf{y}\in S(\mathbf{z})} [\mathbf{a}(\mathbf{y}) + s_{\mathbf{z}}(\mathbf{y})]$.

This means that $-\mathbf{d}(\mathbf{z}) = \bigwedge\limits_{\mathbf{y}\in S(\mathbf{z})} [\mathbf{a}(\mathbf{y}) + s_{\mathbf{z}}(\mathbf{y})] = \mathbf{c}(\mathbf{z})$.

Q.E.D.

Lemma 4.8.25. $(\mathbf{a} \boxtimes \mathbf{s}) \boxtimes \mathbf{t} = \mathbf{a} \boxtimes (\mathbf{s} \boxtimes \mathbf{t})$ for $\mathbf{a} \in \mathbf{R}^{X}$, $\mathbf{t} \in \mathbf{R}_{Y\to Z}$, $\mathbf{s} \in \mathbf{R}_{Z\to X}$.

Proof: Let $\mathbf{b}$ be the image $\mathbf{b} = (\mathbf{a} \boxtimes \mathbf{s}) \boxtimes \mathbf{t}$, and let $\mathbf{c} = \mathbf{a} \boxtimes (\mathbf{s} \boxtimes \mathbf{t})$. Then

$$\mathbf{b}(\mathbf{y}) = \bigvee_{\mathbf{z}\in T(\mathbf{y})} \left\{ \left(\bigvee_{\mathbf{x}\in S(\mathbf{z})} \mathbf{a}(\mathbf{x}) + s_{\mathbf{z}}(\mathbf{x}) \right) + t_{\mathbf{y}}(\mathbf{z}) \right\} = \bigvee_{\mathbf{z}\in T(\mathbf{y})} \left\{ \bigvee_{\mathbf{x}\in S(\mathbf{z})} \mathbf{a}(\mathbf{x}) + s_{\mathbf{z}}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{z}) \right\} =$$

$\bigvee\limits_{\mathbf{z}\in T(\mathbf{y})} \{\mathbf{a}(\mathbf{x}) + s_{\mathbf{z}}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{z}) : \mathbf{x}\in S(\mathbf{z})\}$. Also, we know that

$$\mathbf{c}(\mathbf{y}) = \bigvee_{\mathbf{w}\in \bigcup\limits_{\mathbf{u}\in T(\mathbf{y})} S(\mathbf{u})} \left\{ \mathbf{a}(\mathbf{w}) + \left(\bigvee_{\mathbf{p}\in T(\mathbf{y})} \{t_{\mathbf{y}}(\mathbf{p}) + s_{\mathbf{p}}(\mathbf{w}) : \mathbf{w}\in S(\mathbf{p})\} \right) \right\} =$$

$$\bigvee_{\mathbf{w}\in \bigcup\limits_{\mathbf{u}\in T(\mathbf{y})} S(\mathbf{u})} \left\{ \bigvee_{\mathbf{p}\in T(\mathbf{y})} \{\mathbf{a}(\mathbf{w}) + t_{\mathbf{y}}(\mathbf{p}) + s_{\mathbf{p}}(\mathbf{w}) : \mathbf{w}\in S(\mathbf{p})\} \right\}.$$ This last equality is true

because if $\mathbf{w} \notin S(\mathbf{p})$ then $\mathbf{a}(\mathbf{w})$ is not added to $t_{\mathbf{y}}(\mathbf{p}) + s_{\mathbf{p}}(\mathbf{w})$. Let B =

$\bigcup\limits_{\mathbf{z}\in T(\mathbf{y})} \{\mathbf{a}(\mathbf{x}) + s_{\mathbf{z}}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{z}) : \mathbf{x}\in S(\mathbf{z})\}$, and let C =

$\bigcup_{w \in \bigcup_{u \in \mathcal{T}(y)} \mathcal{S}(u)} \left\{ \bigcup_{p \in \mathcal{T}(y)} \{a(w) + t_y(p) + s_p(w) : w \in \mathcal{S}(p)\} \right\}$ Choose an element in B. It is of form $a(x) + s_z(x) + t_y(z)$, where $z \in \mathcal{T}(y)$ and $x \in \mathcal{S}(z)$. Since $\mathcal{S}(z) \subset \bigcup_{u \in \mathcal{T}(y)} \mathcal{S}(u)$, we have $x \in \bigcup_{u \in \mathcal{T}(y)} \mathcal{S}(u)$. Let $w = x$ and $p = z$. Then $a(x) + s_z(x) + t_y(z) = a(w) + s_p(w) + t_y(p) \in C$. Therefore, $B \subset C$. Conversely, choose an element $a(w) + t_y(p) + s_p(w)$ in C. Then $p \in \mathcal{T}(y)$ and $w \in \mathcal{S}(p)$. Let $x = w$ and $z = p$. Then $a(w) + t_y(p) + s_p(w) = a(x) + s_z(x) + t_y(z) \in B$ and thus $C \subset B$. Therefore, $C = B$, and we are done.

Q.E.D.

Corollary 4.8.26. $(a \boxtimes s) \boxtimes t = a \boxtimes (s \boxtimes t)$, for $a \in R^X$, $t \in R_{Y \to Z}$, $s \in R_{Z \to X}$.

Theorem 4.8.27. $(a \boxtimes -s) \boxtimes -t = a \boxtimes [-(s \boxtimes t)]$ for $a \in R^X$, $t \in R_{Y \to Z}$, $s \in R_{Z \to X}$.

Proof: By Corollary 4.8.26, we know $(a \boxtimes -s) \boxtimes -t = a \boxtimes (-s \boxtimes -t)$, and by Lemma 4.8.1, we know $-s \boxtimes -t = -(s \boxtimes t)$. Thus, $(a \boxtimes -s) \boxtimes -t = a \boxtimes [-(s \boxtimes t)]$.

Q.E.D.

Theorem 4.8.28. $(a \boxtimes s) \boxtimes t \leq (a \boxtimes t) \boxtimes s$, s, t circulant, and $a \in R^X$, $t \in R_{Y \to Z}$, $s \in R_{Z \to X}$.

Proof: Let $d = (a \boxtimes s) \boxtimes t$ and let $e = (a \boxtimes t) \boxtimes s$. The gray values of d and e at $y \in Y$ are, respectively,

$$d(y) = \bigvee_{z \in \mathcal{T}(y)} \left\{ \bigwedge_{x \in \mathcal{S}(z)} \{a(x) + s_z(x) + t_y(z)\} \right\}$$

$$e(y) = \bigwedge_{w \in \mathcal{S}(y)} \left\{ \bigvee_{u \in \mathcal{T}(w)} \{a(u) + t_w(u) + s_y(w)\} \right\}$$

Let $\mathcal{T}(y) = \{z_1, \ldots, z_n\}$, and let $\mathcal{S}(z_i) = \{x_{i1}, \ldots, x_{ik}\}$. Also let $\mathcal{S}(y) = \{w_1, \ldots, w_k\}$, and let $\mathcal{T}(w_i) = \{u_{i1}, \ldots, u_{in}\}$. Note that $|\mathcal{S}(p)| = k$, and $|\mathcal{T}(p)| = n \ \forall \ p \in X$, as s and t are circulant templates. Now we can write the gray values of d and e as

$$d(y) = \bigvee_{i=1}^{n} \left\{ \bigwedge_{j=1}^{k} \{a(x_{ij}) + s_{z_i}(x_{ij}) + t_y(z_i)\} \right\}$$

$$\mathbf{e}(\mathbf{y}) = \bigwedge_{p=1}^{k} \left\{ \bigvee_{q=1}^{n} \{\mathbf{a}(\mathbf{u}_{pq}) + t_{\mathbf{w}_p}(\mathbf{u}_{pq}) + s_{\mathbf{y}}(\mathbf{w}_p)\} \right\}$$

Letting $\delta_{ij} = \mathbf{a}(\mathbf{x}_{ij}) + s_{\mathbf{z}_i}(\mathbf{x}_{ij}) + t_{\mathbf{y}}(\mathbf{z}_i)$ and $\epsilon_{pq} = \mathbf{a}(\mathbf{u}_{pq}) + t_{\mathbf{w}_p}(\mathbf{u}_{pq}) + s_{\mathbf{y}}(\mathbf{w}_p)$, we can write

$$\mathbf{d}(\mathbf{y}) = \bigvee_{i=1}^{n} \left\{ \bigwedge_{j=1}^{k} \delta_{ij} \right\}$$

$$\mathbf{e}(\mathbf{y}) = \bigwedge_{p=1}^{k} \left\{ \bigvee_{q=1}^{n} \epsilon_{pq} \right\}$$

Since $|\{\delta_{ij}\}|_{i=1\ \ j=1}^{n\ \ k} = |\{\epsilon_{pq}\}|_{p=1\ \ q=1}^{k\ \ n}$, (as the templates are circulant), if we can show that $\{\delta_{ij}\} \subset \{\epsilon_{pq}\}$, then we will have $\{\delta_{ij}\} = \{\epsilon_{pq}\}$. Using a theorem of Birkhoff's called the min-max property, which states $\bigvee_{j=1}^{n} \bigwedge_{i=1}^{m} a_{ij} \leq \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} a_{ij}$, Reference 28, we will have our result as desired. To this end, let $\delta_{ij} = \mathbf{a}(\mathbf{x}_{ij}) + s_{\mathbf{z}_i}(\mathbf{x}_{ij}) + t_{\mathbf{y}}(\mathbf{z}_i)$. Since $\mathbf{s}$ is circulant, it is translation invariant, and we have $\mathbf{y} = <\mathbf{z}_i + \mathbf{v}>$, for some $\mathbf{v} \in \mathbf{Z}^k$. Thus $<\mathbf{y} - \mathbf{v}> = \mathbf{z}_i$. Let $\mathbf{m} = <\mathbf{x}_{ij} + \mathbf{v}>$. Note that $<\mathbf{y} + \mathbf{x}_{ij} + \mathbf{v} - \mathbf{y}> = <\mathbf{x}_{ij} + \mathbf{v}>$ $= \mathbf{m}$. *Therefore*, $t_{\mathbf{y}}(\mathbf{z}_i) = t_{<\mathbf{y}+\mathbf{x}_{ij}+\mathbf{v}-\mathbf{y}>}(<\mathbf{z}_i + \mathbf{x}_{ij} + \mathbf{v} - \mathbf{y}>) =$
$t_{\mathbf{m}}(<\mathbf{z}_i + \mathbf{x}_{ij} + \mathbf{v} - \mathbf{y}>) = t_{\mathbf{m}}(<\mathbf{y} - \mathbf{v} + \mathbf{x}_{ij} + \mathbf{v} - \mathbf{y}>) = t_{\mathbf{m}}(<\mathbf{x}_{ij}>) = t_{\mathbf{m}}(\mathbf{x}_{ij})$. Since $\mathbf{t}$ is circulant, we have $\mathbf{x}_{ij} \in \mathcal{T}(\mathbf{m})$. Also, $s_{\mathbf{z}_i}(\mathbf{x}_{ij}) = s_{<\mathbf{z}_i+\mathbf{v}>}(<\mathbf{x}_{ij} + \mathbf{v}>) = s_{\mathbf{y}}(\mathbf{m})$.
Again, $\mathbf{s}$ circulant gives us that $\mathbf{m} \in \mathcal{S}(\mathbf{y})$. Therefore,
$\mathbf{a}(\mathbf{x}_{ij}) + s_{\mathbf{z}_i}(\mathbf{x}_{ij}) + t_{\mathbf{y}}(\mathbf{z}_i) = \mathbf{a}(\mathbf{x}_{ij}) + t_{\mathbf{m}}(\mathbf{x}_{ij}) + s_{\mathbf{y}}(\mathbf{m})$, where $\mathbf{m} \in \mathcal{S}(\mathbf{y})$ and $\mathbf{x}_{ij} \in \mathcal{T}(\mathbf{m})$.
Letting $\mathbf{w}_p = \mathbf{m}$, we have for some q, $\mathbf{u}_{pq} = \mathbf{x}_{ij}$. Thus
$\mathbf{a}(\mathbf{x}_{ij}) + t_{\mathbf{m}}(\mathbf{x}_{ij}) + s_{\mathbf{y}}(\mathbf{m}) = \mathbf{a}(\mathbf{u}_{pq}) + t_{\mathbf{w}_p}(\mathbf{u}_{pq}) + s_{\mathbf{y}}(\mathbf{w}_p)$. Thus we know $\{\delta_{ij}\} \subset \{\epsilon_{pq}\}$.
Applying the min-max property, we are done.

Q.E.D.

Lemma 4.8.29. $(\mathbf{a} \boxtimes \mathbf{s}) \boxtimes \mathbf{t} = (\mathbf{a} \boxtimes \mathbf{t}) \boxtimes \mathbf{s}$, $\mathbf{s}$, $\mathbf{t}$ circulant and $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$,
$\mathbf{s},\mathbf{t} \in \mathbf{R}_{\mathbf{X}\to\mathbf{X}}$.

Proof: We know that $(\mathbf{a} \boxtimes \mathbf{s}) \boxtimes \mathbf{t} = \mathbf{a} \boxtimes (\mathbf{s} \boxtimes \mathbf{t})$. Since $\mathbf{s}$ and $\mathbf{t}$ are circulant, we have $\mathbf{s} \boxtimes \mathbf{t} = \mathbf{t} \boxtimes \mathbf{s}$. Thus, $\mathbf{a} \boxtimes (\mathbf{s} \boxtimes \mathbf{t}) = \mathbf{a} \boxtimes (\mathbf{t} \boxtimes \mathbf{s})$.

Q.E.D.

Corollary 4.8.30. $(\mathbf{a} \boxtimes \mathbf{s}) \boxtimes \mathbf{t} = (\mathbf{a} \boxtimes \mathbf{t}) \boxtimes \mathbf{s}$, $\mathbf{s}$, $\mathbf{t}$ circulant and $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, $\mathbf{s},\mathbf{t} \in \mathbf{R}_{\mathbf{X}\to\mathbf{X}}$.

Lemma 4.8.31. $(\mathbf{a} \vee \mathbf{b}) \boxtimes \mathbf{t} = (\mathbf{a} \boxtimes \mathbf{t}) \vee (\mathbf{b} \boxtimes \mathbf{t})$ $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, $\mathbf{t} \in \mathbf{R}_{\mathbf{Y}\to\mathbf{X}}$.

Proof: On the left hand side of the equation we have for the gray value at $\mathbf{y} \in \mathbf{Y}$,

$$\bigvee_{\mathbf{x}\in T(\mathbf{y})} [\mathbf{a}(\mathbf{x}) \vee \mathbf{b}(\mathbf{x})] + t_{\mathbf{y}}(\mathbf{x}),$$

and on the right hand side we have

$$\bigvee_{\mathbf{x}\in T(\mathbf{y})} \{\mathbf{a}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x})\} \quad \vee \quad \left(\bigvee_{\mathbf{x}\in T(\mathbf{y})} \{\mathbf{b}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x})\}\right).$$

But $\bigvee_{\mathbf{x}\in T(\mathbf{y})} [\mathbf{a}(\mathbf{x}) \vee \mathbf{b}(\mathbf{x})] + t_{\mathbf{y}}(\mathbf{x}) = \left\{\bigvee_{\mathbf{x}\in T(\mathbf{y})} [\mathbf{a}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x})]\right\} \vee \left\{\bigvee_{\mathbf{x}\in T(\mathbf{y})} [\mathbf{b}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x})]\right\} =$

$$[\bigvee_{\mathbf{x}\in T(\mathbf{y})} \{\mathbf{a}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x})\}] \vee \left(\bigvee_{\mathbf{x}\in T(\mathbf{y})} \{\mathbf{b}(\mathbf{x}) + t_{\mathbf{y}}(\mathbf{x})\}\right).$$

Q.E.D.

Corollary 4.8.32. $(\mathbf{a} \wedge \mathbf{b}) \boxtimes \mathbf{t} = (\mathbf{a} \boxtimes \mathbf{t}) \wedge (\mathbf{b} \boxtimes \mathbf{t})$ $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, $\mathbf{t} \in \mathbf{R}_{\mathbf{Y}\to\mathbf{X}}$.

Lemma 4.8.33. $\mathbf{a} \geq (\mathbf{a} \boxtimes -\mathbf{t}) \boxtimes \mathbf{t}'$, where $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}_{\geq 0}$, $\mathbf{t} \in \mathbf{R}^{\geq 0}_{\mathbf{X}\to\mathbf{X}}$, and $\mathbf{t}'$ is the reflection of $\mathbf{t}$.

The proof of this is similar to the case involving $\oslash$ (Lemma 4.7.42).

Corollary 4.8.34. $\mathbf{a} \leq (\mathbf{a} \boxtimes \mathbf{t}') \boxtimes -\mathbf{t}$,where $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}_{\geq 0}$, $\mathbf{t} \in \mathbf{R}^{\geq 0}_{\mathbf{X}\to\mathbf{X}}$.

These next few properties involve $+$, $\vee$, and $\boxtimes$.

Lemma 4.8.35. $\frac{1}{2}(\mathbf{a} \boxtimes \mathbf{s} + \mathbf{a} \boxtimes \mathbf{t}) \leq \mathbf{a} \boxtimes (\mathbf{s} + \mathbf{t})$, $\mathbf{a} \in \mathbf{R}^{\mathbf{X}}$, $\mathbf{s}$, $\mathbf{t} \in \mathbf{R}^{\geq 0}_{\mathbf{Y}\to\mathbf{X}}$.

Proof: At location $\mathbf{y} \in \mathbf{Y}$, $\mathbf{a} \boxtimes \mathbf{s} + \mathbf{a} \boxtimes \mathbf{t}$ has gray value

$$\left(\bigvee_{\mathbf{x}\in S(\mathbf{y})} [\mathbf{a}(\mathbf{x}) + s_{\mathbf{y}}(\mathbf{x})]\right) + \left(\bigvee_{\mathbf{z}\in T(\mathbf{y})} [\mathbf{a}(\mathbf{z}) + t_{\mathbf{y}}(\mathbf{z})]\right),$$

and $\mathbf{a} \boxtimes (\mathbf{s} + \mathbf{t})$ has gray value

$$\bigvee_{\mathbf{w}\in T(\mathbf{y})\cup S(\mathbf{y})} \mathbf{a}(\mathbf{w}) + [s_{\mathbf{y}}(\mathbf{w}) + t_{\mathbf{y}}(\mathbf{w})]$$

Now $\bigvee_{\mathbf{w}\in T(\mathbf{y})\cup S(\mathbf{y})} \mathbf{a}(\mathbf{w}) + [s_{\mathbf{y}}(\mathbf{w}) + t_{\mathbf{y}}(\mathbf{w})] \geq \bigvee_{\mathbf{w}\in T(\mathbf{y})\cup S(\mathbf{y})} \mathbf{a}(\mathbf{w}) + [s_{\mathbf{y}}(\mathbf{w})] \geq$

$\bigvee_{\mathbf{w}S(\mathbf{y})} \mathbf{a}(\mathbf{w}) + [s_{\mathbf{y}}(\mathbf{w})]$, as $s_{\mathbf{y}}(\mathbf{w}) \geq 0$ and $S(\mathbf{y}) \subset T(\mathbf{y})\cup S(\mathbf{y})$. Similarly,

$\bigvee_{\mathbf{w}\in T(\mathbf{y})\cup S(\mathbf{y})} \mathbf{a}(\mathbf{w}) + [s_{\mathbf{y}}(\mathbf{w}) + t_{\mathbf{y}}(\mathbf{w})] \geq \bigvee_{\mathbf{w}T(\mathbf{y})} \mathbf{a}(\mathbf{w}) + [t_{\mathbf{y}}(\mathbf{w})]$. Thus,

$$2*\left[\bigvee_{w\in T(y)\cup S(y)} a(w)+[s_y(w)+t_y(w)]\right]\geq\left[\bigvee_{wS(y)} a(w)+s_y(w)\right]+\left[\bigvee_{wT(y)} a(w)+t_y(w)\right]$$

Therefore

$$2*[a\boxtimes(s+t)]\ \geq\ (a\boxtimes s)+(a\boxtimes t)$$

and our result follows.

Q.E.D.

Corollary 4.8.36. $\frac{1}{2}(a\boxtimes s+a\boxtimes t)\leq a\boxtimes(s+t)$, $a\in R^X$, $s,t\in R^{\geq 0}_{Y-X}$.

c. The Division Algorithm for $\boxtimes$

Given an image $a\in R^X_{\geq 0}$ and an invariant template $t\in R^{\geq 0}_{X-X}$ where $x\in T(x)$, then there exist images q and r such that

$$a=q\boxtimes t'\vee r$$

In other words, an expression similar to the division algorithm ($a=q*t+r$) holds for a and t under $\boxtimes$ as well.

It has been shown that choosing $q=a\boxtimes -t$, and $r=a*[\chi_{>0}(a-[(a\boxtimes -t)\boxtimes t'])]$ will satisfy the above equation.

As was the case for $\oslash$, we can make the following recursive definitions and write a in series form.

$$a=r_0\vee(r_1\boxtimes t')\vee(r_2\boxtimes(t')^2)\vee\cdots\vee(r_{n-1}\boxtimes(t')^{n-1})\vee(a_n\boxtimes(t')^n)$$

$$=\left\{\bigvee_{k=0}^{n-1} r_k\boxtimes(t')^k\right\}\vee(a_n\boxtimes(t')^n)$$

where

$a_0\equiv a$,

$r_k=a_k*[\chi_{>0}(a_k-[(a_k\boxtimes -t)\boxtimes t'])]$, $k=0,\ldots,n-1$,

$a_k=a_{k-1}\boxtimes -t$, $k=1,\ldots,n$,

and

$(t')^k \equiv t' \boxtimes t' \cdots \boxtimes t'$, k times.

Note that a_k must be non-negative for each k=1,...,n, in order to continue to apply the division algorithm to it.

It can be shown, Reference 29, that for each $k = 0,1,2,...,$

$$v((r_k \boxtimes -t) \boxtimes t') \leq v(t),$$

where v is the valuation function defined as above. Thus in this sense each remainder r_k is small.

In the boolean case, Reference 30, there exists an n such that

$$a_n \boxtimes (t')^n = 0$$

so that the expression for **a** becomes

$$a = \bigvee_{k=0}^{n-1} r_k \boxtimes (t')^k$$

One useful application of this result for the boolean case is in data compression. By encoding the r_i's in run length code, the image can be represented by fewer bits of data, and reconstructed exactly once **t** is known.

9. IMAGE ALGEBRA AND MATHEMATICAL MORPHOLOGY

This section discusses the connection between the two structures of the Image Algebra under $\bigvee$ and $\boxtimes$ and mathematical morphology, in particular how these structures generalize morphological concepts. Mathematical morphology, first pioneered by G. Matheron, Reference 31, and J. Serra, Reference 32, depends on two fundamental set theoretic operations, namely the Minkowski addition and subtraction of sets in Euclidean space, Reference 33. For any set $A \subset \mathbf{R}^n$ and $B \subset \mathbf{R}^n$, Minkowski addition and subtraction are defined as

$$A \oplus B = \{a+b : a \in A,\ b \in B\} \text{ and } A \ominus B = (A^c \oplus B')^c ,$$

respectively, where $B' = \{ -b : b \in B \}$ and A^c denotes the complement of A in $\mathbf{R}^n$. The two basic morphological operations of erosion and dilation of A by B are then defined as $A \ominus B'$ and $A \oplus B'$, respectively. To avoid anomalies without practical interest, it is usually assumed that B contains the origin. In applications A represents a set of pixels and B is referred to as a structuring element.

A more general image transform, called the Hit or Miss transform, can be defined in terms of the Minkowski subtraction. This transform is often viewed as the universal morphological transform upon which the theory of mathematical morphology is based. Its definition is as follows. Suppose B = (D,E) is a pair of structuring elements, then the Hit or Miss transform of the set A is given by the expression

$$A \circledast B = \{a : D_a \subset A,\ E_a \subset A^c\}$$

where D_a denotes the translate $D + a$ of D by a, $a \in A$, and similarly for E_a. For practical applications, it is assumed that $D \cap E = \emptyset$. It turns out that if $E = \emptyset$, then $A \circledast B = A \ominus D'$. Thus, the Hit or Miss transform includes erosion and, hence, dilation as a special case.

Of course, in actual image processing we are not dealing with arbitrary sets in Euclidean space but bounded arrays of fixed size. Thus, in the above definitions we cannot simply substitute an image for the symbol A, as the expression A^c would become meaningless to the computer. What is usually assumed is that A is a set of points within a given image such as the set of all black pixels in a Boolean image. Another observation is that the Minkowski operations are not applied in the same manner to gray valued images as they are to Boolean images. In fact, the operations of erosion and dilation for gray valued images have to be redefined through the cumbersome notion of the umbra of an image, Reference 32. In contrast, the simple max and min Image Algebra convolutions defined by $\boxtimes$ and $\boxtimes$ can be used to express the morphological operations of dilation and erosion, respectively, for both Boolean and gray valued images. In particular, if B denotes the structuring element, we may define a template $\mathbf{t} = (T, t)$ corresponding to B by setting $T(\mathbf{y}) = B'_{\mathbf{y}}$, and let $\mathbf{t}_{\mathbf{y}}(\mathbf{x})$ correspond to the values assigned to $B_{\mathbf{y}}'$ at location $\mathbf{x}$ with $\mathbf{t}_{\mathbf{y}}(\mathbf{x}) = 0$ if $\mathbf{x} \notin T(\mathbf{y})$. Then,

$\mathbf{a} \boxtimes \mathbf{t}$ is equivalent to a dilation of A by B, and

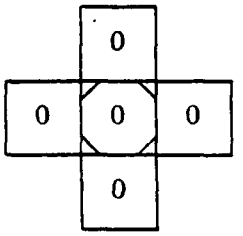
$\mathbf{a} \boxtimes -\mathbf{t}$ is equivalent to an erosion of A by B, where $-\mathbf{t} = (T, -t)$.

A little care needs to be taken in the interpretation of the above two statements. For instance, in the Image Algebra expression $\mathbf{a} \boxtimes \mathbf{t}$, $\mathbf{a}$ denotes the actual input image. Thus, for Boolean image processing ($\mathbf{t}_{\mathbf{y}}(\mathbf{x}) = 0\ \forall\ \mathbf{x}$), the output image $\mathbf{a} \boxtimes \mathbf{t}$ will have the black components of $\mathbf{a}$ ($a(\mathbf{x}) = 1$) dilated. In contrast, in the morphological transform $A \oplus B'$, A does not denote the input image but only the set of black pixels of the input image, while $A \oplus B'$ denotes the set of black pixels of the output image. In addition, structuring elements correspond only to invariant templates from $\mathbf{X}$ to $\mathbf{X}$ with the property that the target pixel

is an element of the source configuration $\check{T}(\mathbf{y})$, a very limited class of templates.

If $\mathbf{t}$ has only zero weights, then $\mathbf{t} = -\mathbf{t}$ and $\mathbf{a} \boxtimes -\mathbf{t} = \mathbf{a} \boxtimes \mathbf{t}$. Hence the eroded image $\mathbf{a} \boxtimes -\mathbf{t}$ is the image obtained from $\mathbf{a}$ by replacing each pixel $a(\mathbf{x})$ by the minimum value in the source configuration (window) $\check{T}(\mathbf{y})$. That this type of min and max operations on Boolean images is equivalent to erosions and dilations follows from P. Miller's Boolean algebra approach to morphology, Reference 30. Figure 8 represents the dilation $\mathbf{a} \boxtimes \mathbf{t}$ of the input image $\mathbf{a}$ shown in Figure 8(a), while Figure 8(c) represents the erosion of the image $\mathbf{a} \boxtimes \mathbf{t}$, namely, $(\mathbf{a} \boxtimes \mathbf{t}) \boxtimes \mathbf{t}$ which is also called the closing of $\mathbf{a}$ by $\mathbf{t}$. (Serra, Reference 32). The template $\mathbf{t}$ used in this example is defined by

$\mathbf{t}(\mathbf{x}) =$



The operational equivalence between $\boxtimes$, $\boxtimes$ and dilations and erosions for gray valued images stems from the fact that when actually implementing dilations and erosions for image processing purposes, the two equations

$$c(\mathbf{x}) = \bigvee_{\mathbf{y}} [a(\mathbf{x}-\mathbf{y}) + b(-\mathbf{y})]$$

$$c(\mathbf{x}) = \bigwedge_{\mathbf{y}} [a(\mathbf{x}-\mathbf{y}) - b(-\mathbf{y})]$$

are used, where $a(\mathbf{x})$ denotes the input pixels and $b(\mathbf{y})$ the values of the structuring element B (Serra, pg. 441, Reference 32).

In the Boolean case it is a simple exercise to show that the Hit or Miss transform can be expressed as

$$A \circledast B = (A \ominus D') \cap (A^c \ominus E').$$

Defining $\mathbf{t} = (\check{T}, t)$ by $\check{T}(\mathbf{y}) = D_{\mathbf{x}}'$, $t_{\mathbf{y}}(\mathbf{x}) = 0$ and $\mathbf{s} = (S, s)$ by $S(\mathbf{y}) = E_{\mathbf{x}}'$ and $\mathbf{s}_{\mathbf{y}}(\mathbf{x}) = 0$, we obtain the equivalent Image Algebra expression

$$(\mathbf{a} \boxtimes \mathbf{t}) * (\bar{\mathbf{a}} \boxtimes \mathbf{s}).$$

However, there is an even simpler Image Algebra formulation of the Hit or Miss transform which does not employ the notions of minimum or erosion. Suppose $\mathcal{T}(\mathbf{y}) = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ and $\mathcal{S}(\mathbf{y}) = \{\mathbf{x}_{k+1}, \ldots, \mathbf{x}_n\}$. Then

$$\chi_m(\mathbf{a} \oplus \mathbf{r}), \text{ where } m = \sum_{i=0}^{k} 2^{i-1},$$

is equivalent to computing the Hit or Miss transform, where $\mathbf{r} = (\mathcal{R}, r)$ is defined by $\mathcal{R}(\mathbf{y}) = \mathcal{T}(\mathbf{y}) \cup \mathcal{S}(\mathbf{y})$ and $\mathbf{r}_y(\mathbf{x}_i) = 2^{i-1}$.

Since Image Algebra can express the two basic morphological operations of erosion and dilation, it is clear that the subalgebra $\mathcal{A} = (\mathbf{R}^X, \mathbf{R}_{X \to X}, +, *, \vee, \boxtimes)$ of the full Image Algebra includes mathematical morphology as a special case. However, even the subalgebra $\mathcal{A}$ is a much larger algebra than that provided by the algebra defined by mathematical morphology. Templates are more general objects than structuring elements. They can vary in size, shape and weights from point to point and they can map between different structures and dimensions if we replace $\mathbf{R}$ by $\mathbf{F}$ and $\mathbf{R}_{X \to X}$ by $\mathbf{F}_{Y \to X}$. Thus, an expression of form $\mathbf{a} \boxtimes \mathbf{t}$ can express a far more complex process than a simple dilation. For example, if $\mathbf{a}$ denotes the input image shown in Figure 36 and $\mathbf{t} = (\mathcal{T}, t)$ is defined by $\mathcal{T}(i,j) = \{(x,y)\}$ with

$$\mathbf{t}_{(x,y)}(x,y) = \begin{cases} 1 & \text{if } 0.9 < \dfrac{x^2}{p^2} + \dfrac{y^2}{q^2} < 1.1 \\ & \text{or } 0.9 < \dfrac{x^2}{d^2} - \dfrac{y^2}{q^2} < 1.1 \\ 0 & \text{otherwise} \end{cases}$$

where q=15, $p^2 = q^2 + c^2$, $d^2 = c^2 - q^2$ and $c = 30$, then $\mathbf{a} \boxtimes \mathbf{t}$ is obviously not a dilation nor is it expressible in terms of dilations and erosions when starting with the input image $\mathbf{a}$. The input and output images are shown in the next two figures.

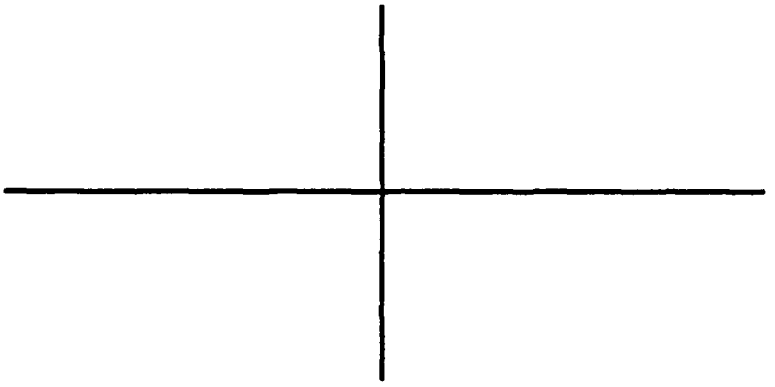


Figure 36. Input Image **a**

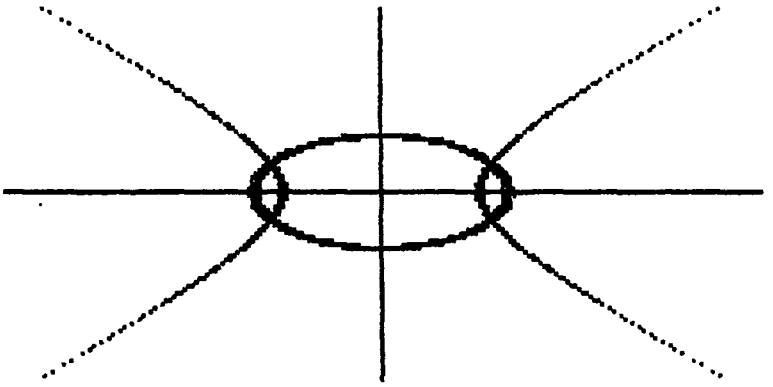


Figure 37. Image $\mathbf{a} \boxtimes \mathbf{t}$

In the Boolean case, morphological operators can be accomplished with either $\oslash$ or $\boxtimes$ by observing that for a Boolean image **a** and a template **t** with $\mathbf{t}_y(\mathbf{x}) = 0 \; \forall \; \mathbf{x} \in T(\mathbf{y})$, $\mathbf{a} \boxtimes \mathbf{t} = \mathbf{a} \oslash \tilde{\mathbf{t}}$, where $\tilde{\mathbf{t}} = (T, \tilde{t})$ and $\tilde{t}_y(\mathbf{x}) = 1$ if $\mathbf{x} \in T(\mathbf{y})$.

SECTION V
IMAGE ALGEBRA SOFTWARE DEVELOPMENT

The goal of the Image Processing Language Project is the development of an algebra suitable for describing image processing transformations. A successful Image Algebra can serve as a direct representation of image processing algorithms if embedded into a programming language. Since the early part of Phase I of the contract, we have been conducting work in the implementation of the Image Algebra in FORTRAN. In this section, we give a brief chronology of that work, as well as other work in software development that we have carried out during the duration of this contract.

A preprocessor translates a source program in an extended version of some language L into a program in L. Since many image processing application programs are written in FORTRAN, an Image Algebra FORTRAN (IAF) preprocessor was undertaken in the Spring of 1984 as a senior project by W.K. Perry. The IAF preprocessor translates FORTRAN programs with image algebra extensions into standard FORTRAN programs. This gave users the ability to define translation invariant templates and to execute binary image-image and image-template operations using a notation similar to the Image Algebra. Various ASCII characters (such as |, #, ^, etc.) were used in this original version of Image Algebra FORTRAN to denote the Image Algebra symbols like $\oplus, \oslash$, and $\boxtimes$, that do not appear on a standard keyboard.

During the Fall of 1985, work on the preprocessor led to the development of a notation for defining translation variant templates. This required a substantial change to the design of the preprocessor. A new program unit, the template definition, was added to the Image Algebra Fortran language.

Until the Fall of 1985, we had been using a Printronix line printer as our primary image output device. Our gray-toning software permitted us to print images on the dot matrix printer using six by six squares of dots to represent a single pixel in an image. At this time, our department acquired a 300 dot per inch resolution laser printer and we modified our gray-toning software to use the laser printer as an output device. This improved the resolution of our image output by a factor of about 16.

During this period, the IAF preprocessor came into heavy use since the preprocessor had been distributed to a number of companies who had expressed an interest in having this

Image Algebra software tool. It became clear that although the preprocessor was adequate to the expression of both Image Algebra expressions and much of standard FORTRAN, it was not ideal at combining both these tasks within a single program. A number of shortcomings were identified:

(1) Image Algebra expressions could only appear within the context of a FORTRAN assignment statement,

(2) although arbitrarily many image-image operations using FORTRAN arithmetic operations could appear in a single assignment statement, if an image-template operation or non-FORTRAN image-image operation appeared in an assignment statement, it must be the only operation in that assignment statement,

(3) the set of variable names used by the preprocessor could clash with many user defined names,

(4) although the image expression of the form `A + 1` would be acceptable, `1 + A` would not be recognized as an image expression,

(5) an image could appear both on the left hand side of an assignment statement and as an operand of an image-template operation on the right hand side,

(6) no statement involving declaration of arrays or assignments to images could be continued on multiple lines, and

(7) almost no error checking was performed by the preprocessor.

These problems stemmed from the fact that the preprocessor performed a simple kind of pattern matching to drive its translation. This simple pattern matching method is much too weak to handle the subtleties of the FORTRAN syntax. Although various improvements were made to the preprocessor, it retained this simple basis, making the improvement of many of its deficiencies impossible without a complete redesign effort.

By the Spring of 1986, it had become clear to us that the Image Algebra is readily implementable in a programming language and can yield fantastic benefits in the speeding up of the program development process. We had demonstrated the IAF preprocessor several times at program reviews and met with enthusiasm and great interest on the part of practically all of those who had seen its use in the development of code. In preparation for potential military funded development of the Image Algebra in a production quality setting, we contacted the several Ada compiler producers. The companies contacted produced Ada

compilers for a wide variety of machines at that time, their products had good reputations. We felt that it was important to contact Ada compiler producers to determine the feasibility of cooperative development of an Image Algebra Ada compiler, that is, a compiler for Ada that provided the user with Image Algebra operations and operands. Such a cooperative effort with a company having existing Ada expertise could yield a workable Ada product with less lead time than development from the ground up. While all the companies contacted indicated that cooperative work could be carried out, their estimates of price and level of interaction differed. Estimates of cost of release of source code varied from $150,000 to $1,000,000.

During the summer of 1986, we were able to arrange a loan of a Sun 3/160C workstation. At that time, we were able to develop an image display package callable by IAF programs. This gave us the capability of displaying pseudo-color representations of 8-bit imagery during processing. We obtained our own Sun Workstations via a DoD University Research Initiative Program grant in December of 1986 and were then able to use the preprocessor to develop nontrivial example programs since our throughput time was greatly enhanced by the workstation's real-time image display capability.

While the Image Algebra FORTRAN preprocessor provides us with the ability to test concepts of the Image Algebra, it does not provide a rich environment for the testing of Image Algebra implementation concepts. A preprocessor has inherent limitations that affect its usability.

- Runtime efficiency is poor since the language extensions provided by a preprocessor must be implemented in terms of a target high-level language that was deficient in providing the desired extensions in the first place. Hence, the extensions may not be implemented in the most appropriate way on any given machine.
- Translation efficiency in a preprocessor is poor since an extra translation step—extended host language to host language—must be carried out to produce a running program.
- User comprehension is degraded by having to understand one extra interface level. Runtime error messages are often reported with respect to the translated host language version of the program rather than the source language.

Bearing in mind these limitations of the preprocessor approach to language implementation, in the fall of 1986, we decided to study the issues involved in implementing the Image

Algebra in a compiler. A compiler is a program that translates a high level language source program into an executable machine language object program. Such a language implementation does not suffer from the problems cited above for preprocessors. A typical compiler consists of two phases: the front end, which is a machine independent compilation phase translating the source code of the compiled language into an intermediate code; and the back end, which is a machine dependent phase translating the intermediate code into the machine language of the target machine.

One question we addressed at this time was what kind of intermediate code might be appropriate for use in such a compiler. After a review of the relevant compiler literature we decided that since Stanford U-Code had been used in the development of a machine independent global data flow optimizer 34 it might be a suitable intermediate code for the Image Algebra. Global data flow analysis is the study of the flow of data values through programs. Many sorts of code improvement techniques can be performed using information derived from the analysis of the dataflow of a program. Since execution of a single Image Algebra operation typically involves many thousands or even millions of operations on data in a highly structured way, there are many opportunities to exploit global dataflow properties during the translation of source code to machine executable object code.

During this period W.K. Perry undertook initial steps required in the development of a compiler for the C language with Image Algebra extensions 35. The designed compiler consists of three major pieces of code: a front end, translating Image Algebra C into assembly code; an assembler, translating the assembly code into object code; and a machine simulator interpreting the produced object code. We avoided compiling directly to the machine language of any particular machine so that the implementation could be ported to machines other than the one on which it was originally implemented. This work demonstrated that U-Code, designed for a uniprocessor model of computation, is in fact a weak language for implementing Image Algebra operations. A vector variant of U-Code, V-Code, was developed to permit more efficient execution of operations in the compiler-assembler-simulator environment.

Another student began a study of global dataflow based code improvements suggested by the kind of code produced by straightforward implementation of the Image Algebra in a uniprocessor setting 36. Her initial task was to study existing Image Algebra algorithms; determine what sorts of data interdependencies appeared in the generated code; and

determine techniques of code modification that could be mechanically applied to yield code with identical results but faster execution speed. One of the initial algorithms she chose to look at was histogram computation. Suppose **a** is an image. One algorithm for computing the histogram **h** of **a** using Image Algebra operations is the following:

for i **in** $\wedge$**a** to $\vee$**a** **do**
  $\mathbf{h}(i) = \sum(\chi_i(\mathbf{a}))$

Expressed in Image Algebra FORTRAN, we have the following algorithm:

```
      do 10 i= !min a, !max a
            h(i) = !sum ( a == i )
   10 continue
```

This leads to generation of the following FORTRAN code in the current preprocessor:

```
*      do 10 i= !min a, !max a
       XXSI01=A(1,1)
       DO 99000 XXLCV2=1,NCOLS
       DO 99001 XXLCV1=1,NROWS
       IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
       XXSI01=A(XXLCV1,XXLCV2)
       ENDIF
99001  CONTINUE
99000  CONTINUE
       XXSI02=A(1,1)
       DO 99002 XXLCV2=1,NCOLS
       DO 99003 XXLCV1=1,NROWS
       IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
       XXSI02=A(XXLCV1,XXLCV2)
       ENDIF
99003  CONTINUE
99002  CONTINUE
       DO 10 I=XXSI01,XXSI02
*            h(i) = !sum ( a == i )
       XXIND3=1
       DO 99004 XXLCV2=1,NCOLS
       DO 99005 XXLCV1=1,NROWS
       IF (A(XXLCV1,XXLCV2).EQ.I) THEN
       XXI001(XXIND3)=1
       ELSE
       XXI001(XXIND3)=0
       ENDIF
       XXIND3=XXIND3+1
99005  CONTINUE
99004  CONTINUE
       XXSI03=0
       XXIND2=1
       DO 99006 XXLCV2=1,NCOLS
       DO 99007 XXLCV1=1,NROWS
       XXSI03=XXSI03+XXI001(XXIND2)
       XXIND2=XXIND2+1
99007  CONTINUE
99006  CONTINUE
       H(I)=XXSI03

   10  continue
```

One of the optimizations being investigated is that of loop joining. Loop joining consists roughly of concatenating the bodies of loops together if they have the same loop bounds. This can only be done if certain properties of the loops hold that guarantee that the joining will not change the behavior of each of the loops. The code as modified by loop joining is the following:

```
*       do 10 i= !min a, !max a
        XXSI01=A(1,1)
C       Assignment to XXSI02 hoisted up from second loop
        XXSI02=A(1,1)
        DO 99000 XXLCV2=1,NCOLS
        DO 99001 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
        XXSI01=A(XXLCV1,XXLCV2)
        ENDIF
C       Code that follows was body of second loop
        IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
        XXSI02=A(XXLCV1,XXLCV2)
        ENDIF
99001   CONTINUE
99000   CONTINUE
        DO 10 I=XXSI01,XXSI02
*             h(i) = !sum ( a == i )
        XXIND3=1
C       These two assignments hoisted up from second loop
        XXSI03=0
        XXIND2=1
        DO 99004 XXLCV2=1,NCOLS
        DO 99005 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).EQ.I) THEN
        XXI001(XXIND3)=1
        ELSE
        XXI001(XXIND3)=0
        ENDIF
        XXIND3=XXIND3+1
C       code that follows was body of the second loop
        XXSI03=XXSI03+XXI001(XXIND2)
        XXIND2=XXIND2+1
99007   CONTINUE
99006   CONTINUE
        H(I)=XXSI03
   10   continue
```

Noting that the two induction variables XXIND3 and XXIND2 have the same value, we can perform what we call backward code motion to yield the following program:

```
*       do 10 i= !min a, !max a
        XXSI01=A(1,1)
        XXSI02=A(1,1)
        DO 99000 XXLCV2=1,NCOLS
        DO 99001 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
        XXSI01=A(XXLCV1,XXLCV2)
        ENDIF
        IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
        XXSI02=A(XXLCV1,XXLCV2)
        ENDIF
99001   CONTINUE
99000   CONTINUE
        DO 10 I=XXSI01,XXSI02
*             h(i) = !sum ( a == i )
        XXIND3=1
        XXSI03=0
        DO 99004 XXLCV2=1,NCOLS
        DO 99005 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).EQ.I) THEN
        XXI001(XXIND3)=1

C       assignment to XXSI03 was moved back into this if branch
        XXSI03=XXSI03+XXI001(XXIND3)
        ELSE
        XXI001(XXIND3)=0

C       Assignment to XXSI03 was moved back into this if branch
        XXSI03=XXSI03+XXI001(XXIND3)
        ENDIF
        XXIND3=XXIND3+1
99007   CONTINUE
99006   CONTINUE
        H(I)=XXSI03
   10   continue
```

Copy propagation is the removing of assignments to variables whose values are copied via assignment to other variables and never used again. Performing copy propagation, we are left with this program:

```
*       do 10 i= !min a, !max a
       XXSI01=A(1,1)
       XXSI02=A(1,1)
       DO 99000 XXLCV2=1,NCOLS
       DO 99001 XXLCV1=1,NROWS
       IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
       XXSI01=A(XXLCV1,XXLCV2)
       ENDIF
       IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
       XXSI02=A(XXLCV1,XXLCV2)
       ENDIF
99001  CONTINUE
99000  CONTINUE
       DO 10 I=XXSI01,XXSI02
*             h(1) = !sum ( a == 1 )

C      XXIND3 has been removed because it now has no effect
C      on the behavior of the program

C      XXSI03 has been removed since it is used only to directly assign
C      it's value to H(I)
       H(I) = 0
       DO 99004 XXLCV2=1,NCOLS
       DO 99005 XXLCV1=1,NROWS
       IF (A(XXLCV1,XXLCV2).EQ.I) THEN

C      XXI001 has been removed since it is used only to directly assign
C      it's value to H(I).
       H(I)=H(I)+1
       ELSE
       H(I) = H(I)+0
       ENDIF
99007  CONTINUE
99006  CONTINUE
   10  continue
```

We can now modify by simplification of algebraic expressions. This makes the assignment of H(I) + 0 to H(I) a needless operation, thus we can remove the else branch containing this statement, yielding the following code:

```
*       do 10 i= !min a, !max a
        XXSI01=A(1,1)
        XXSI02=A(1,1)
        DO 99000 XXLCV2=1,NCOLS
        DO 99001 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
        XXSI01=A(XXLCV1,XXLCV2)
        ENDIF
        IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
        XXSI02=A(XXLCV1,XXLCV2)
        ENDIF
99001   CONTINUE
99000   CONTINUE
        DO 10 I=XXSI01,XXSI02
*             h(i) = !sum ( a == i )
        H(I) = 0
        DO 99004 XXLCV2=1,NCOLS
        DO 99005 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).EQ.I) THEN
        H(I)=H(I)+1

C       Else branch with no effect removed
        ENDIF
99007   CONTINUE
99006   CONTINUE
   10   continue
```

Loop Splitting consists of breaking the statements of one loop apart and placing them into two loops. This can be useful if it is possible to further optimize one of the loops in the absence of the statements in the other. Following this idea, we split the loop over the gray value range of the image **a**, yielding this version of the program:

```
*       do 10 i= !min a, !max a
        XXSI01=A(1,1)
        XXSI02=A(1,1)
        DO 99000 XXLCV2=1,NCOLS
        DO 99001 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
        XXSI01=A(XXLCV1,XXLCV2)
        ENDIF
        IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
        XXSI02=A(XXLCV1,XXLCV2)
        ENDIF
99001   CONTINUE
99000   CONTINUE

C       This loop has been introduced
        DO 5 I=XXSI01,XXSI02
        H(I) = 0
    5   CONTINUE

        DO 10 I=XXSI01,XXSI02
*             h(1) = !sum ( a == i )
        DO 99004 XXLCV2=1,NCOLS
        DO 99005 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).EQ.I) THEN
        H(I)=H(I)+1
        ENDIF
99007   CONTINUE
99006   CONTINUE
   10   continue
```

Given certain conditions on the body of a nested loop, it is possible to interchange the loop iterations. In this version of the program, we interchange the loop over I with the two loops nested within it:

```
*       do 10 i= !min a, !max a
        XXSI01=A(1,1)
        XXSI02=A(1,1)
        DO 99000 XXLCV2=1,NCOLS
        DO 99001 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
        XXSI01=A(XXLCV1,XXLCV2)
        ENDIF
        IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
        XXSI02=A(XXLCV1,XXLCV2)
        ENDIF
99001   CONTINUE
99000   CONTINUE
        DO 5 I=XXSI01,XXSI02
        H(I) = 0
     5  CONTINUE
*               h(i) = !sum ( a == i )
        DO 99004 XXLCV2=1,NCOLS
        DO 99005 XXLCV1=1,NROWS

C       loop on I has been shifted to be innermost
        DO 10 I=XXSI01,XXSI02
        IF (A(XXLCV1,XXLCV2).EQ.I) THEN
        H(I)=H(I)+1
        ENDIF
    10  continue
99007   CONTINUE
99006   CONTINUE
```

If two consecutive if statements without else branches are mutually exclusive, that is, if given that one of the if tests is satisfied the other is not satisfied and vice versa, then we can join these statements into an if-then-else statement. Likewise, if the body of a do-loop consists of only a single if-statement with no else branch, and the if condition is satisfied only when the loop index takes on a particular value v, then the loop can be removed and all references to the do-loop index within the if can be replaced by v. Performing these modifications results in the following program:

```
*       do 10 i= !min a, !max a
        XXSI01=A(1,1)
        XXSI02=A(1,1)
        DO 99000 XXLCV2=1,NCOLS
        DO 99001 XXLCV1=1,NROWS
        IF (A(XXLCV1,XXLCV2).LT.XXSI01) THEN
        XXSI01=A(XXLCV1,XXLCV2)

C       We join this IF with the previous one by else introduction
        ELSE IF (A(XXLCV1,XXLCV2).GT.XXSI02) THEN
        XXSI02=A(XXLCV1,XXLCV2)
        ENDIF
99001   CONTINUE
99000   CONTINUE
        DO 5 I=XXSI01,XXSI02
        H(I) = 0
    5   CONTINUE
*             h(1) = !sum ( a == 1 )
        DO 99004 XXLCV2=1,NCOLS
        DO 99005 XXLCV1=1,NROWS

C       This statement was guarded by a loop joinable IF
        H(A(XXLCV1,XXLCV2))=H(A(XXLCV1,XXLCV2))+1
99007   CONTINUE
99006   CONTINUE
```

At this point, the example has been modified to the point where it is effectively no different from the sort of algorithm that a programmer might develop by hand for histogram computation. The speedup achieved by this modification is about a factor of 300 on our Sun Microsystems 3/280 uniprocessor. We are continuing our study of code improvement techniques, trying to identify necessary and sufficient conditions to insure that the modified code will run faster than the original code. We are also searching for ways to decrease the amount of time needed to determine the best restructuring of the original code.

It is our hope that this seminal work on implementation of the Image Algebra can serve as a starting point for the development of production quality software products providing computer vision program developers with the capability to solve image processing tasks with directly implemented Image Algebra primitives. Our experience has shown that not only is it possible to enhance the productivity of programmers using the Image Algebra, but it is also possible to efficiently implement Image Algebra primitives in a programming language system.

REFERENCES

1. P.E. Miller, "Development of a Mathematical Structure for Image Processing," Optical Division Tech. Report, Perkin-Elmer (1983).
2. G.X. Ritter, et. al., "A Synopsis of Common Image Transforms and Techniques," TR #1, Image Algebra Project, FO8635-84-C-0295, Eglin AFB, FL (1985).
3. G.X. Ritter, et. al., "Program Review #2, Viewgraphs and Notes," Image Algebra Project, FO8635-84-C-0295, Eglin AFB, FL (March 27, 1985).
4. G.X. Ritter, et. al., "Program Review #3, Viewgraphs and Notes," Image Algebra Project, FO8635-84-C-0295, Eglin AFB, FL (June 11, 1985).
5. G.X. Ritter and P.D. Gader, "Image Algebra Techniques for Parallel Image Processing," *Journal of Parallel and Distributed Computing* 4(5) (March 1987), 7-44.
6. P.D. Gader, "Image Algebra Techniques for Parallel Computation of Discrete Fourier Transforms and General Linear Transforms," Ph.D. Dissertation, University of Florida, Gainesville, FL (1986).
7. J.N. Wilson, et. al., "Artificial Intelligence Feasibility/Augmentation Report," TR (8) Image Algebra Project, F08635-84-C-0295, Eglin AFB, FL (1987).
8. G. Ritter and J. Wilson, "Image Algebra: A New Approach to Algorithm Development," in *IEEE Computer Society 16th AIPR Workshop*, Washington, D.C. (October, 1987).
9. G.X. Ritter, et. al., "Standard Image Processing Algebra Document Phase II," TR (7) Image Algebra Project, F08635-84-C-0295, Eglin AFB, FL (1987).
10. G.X. Ritter, M.A. Shrader-Frechette, and J.N. Wilson, "Image Algebra: A Rigorous and Translucent Way of Expressing All Image Processing Operations," in *Proc. of the 1987 SPIE Tech. Symp. Southeast on Optics, Elec.-Opt., and Sensors*, Orlando, FL (May 1987).
11. G.X. Ritter and P.D. Gader, "Image Algebra Implementation on Cellular Array Computers," pp. 430-438 in *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL (1985).
12. G.X. Ritter and J.N. Wilson, "Image Algebra: A Unified Approach to Image Processing," in *Proceedings of the SPIE Medical Imaging Conference*, Newport Beach, CA (February 1987).
13. G.X. Ritter, J.L. Davidson, and J.N. Wilson, "Beyond Mathematical Morphology," pp. 260-269 in *Proc. of SPIE Conf. - Visual Communication and Image Processing II*, Cambridge, MA (October 1987).
14. A. Rosenfeld, "Parallel Image Processing Using Cellular Array Computers," *Computer* 1(1) (January 1983), 177-191.
15. D. Marr and E. Hildreth, "Theory of Edge Detection," *Proc. R. Soc. London B* **207** (1980).
16. E. Kaltofen, "Polynomial Time Reduction from Multivariate to Bivariate to Univariate Integer Polynomial Factorization," *SIAM J. Comput* **14** (1985), 469-489.
17. D.R. Musser, "Multivariate Polynomial Factorization," *Journal of the ACM* **22**(2) (April 1975), 291-308.
18. S. Winograd, "On Computing the Discrete Fourier Transform," *Math. Comp.* **32** (January 1978), 175-199.

19. R. Blahut, *Fast Algorithms for Digital Signal Processing,* Addison-Wesley, Reading, MA (1985).

20. H.J. Nussbaumer and P. Quandelle, "Fast Computation of Discrete Fourier Transforms using Polynomial Transforms," *IEEE Transactions Acoust., Speech, Signal Proc.* **ASSP-27** (1979).

21. M.C. Pease, "An Adaptation of the Fast Fourier Transform for Parallel Processing," *Journal ACM* **15**(2) (April 1968), 253-264.

22. C.R. Jesshope, "The Implementation of Fast Radix Two Transforms on Array Processors," *IEEE Transactions on Computers* **C-29**(1) (January 1980), 20-27.

23. J.P. Strong, "The Fourier Transform on Mesh Connected Arrays Such as the Massively Parallel Processor," pp. 190-197 in *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management,* Miami Beach, FL (Nov. 1985).

24. D.J. Rose, "Matrix Identities of the Fast Fourier Transform," *Linear Algebra and Its Applications* **29** (1980), 423-443.

25. M. Tchuente, "Parallel Realization of Permutations Over Trees," *Discrete Math.* **39** (1982), 211-214.

26. J.H. McClellan and C.M. Rader, edit., *Number Theory in Digital Signal Processing,* Prentice-Hall, Englewood Cliffs, NJ (1979).

27. P.D. Gader, "Parallel Algorithms for Computing Discrete Fourier Transforms via Tridiagonal Factorizations of Fourier Matrices," *Preprint,* (1986).

28. Birkhoff, *Lattice Theory,* American Mathematical Society, Providence, RI (1984).

29. G.X. Ritter, J.L. Davidson, "The Image Algebra and Lattice Theory," UF-CIS Technical Report TR-87-09, Dept. of Comp. and Info. Sci., Univ. of Florida, Gainesville, FL (to be published).

30. P.E. Miller, "An Investigation of Boolean Image Neighborhood Transformations," Ph.D. thesis, Ohio State University (1978).

31. G. Matheron, *Random Sets and Integral Geometry,* Wiley, New York (1975).

32. J. Serra, *Image Analysis and Mathematical Morphology,* Academic Press, London (1982).

33. Hadwiger, *Vorlesungen* über Inhalt, Oberflache und Isoperimetrie, Springer-Verlag, Berlin (1957).

34. F.C. Chow, "A Portable Machine-Independent Global Optimizer—Design and Measurements," Ph.D. Dissertation, Stanford University (December 1983).

35. W.K. Perry, "IAC: Image Algebra C," Masters Thesis, University of Florida CIS Department (August 1987).

36. L.A. White, "Extension of Global Data Flow Optimizations to Image Processing," Ph.D. Dissertation Proposal, University of Florida CIS Department (October 1987).